Finding Digital Convexity

Loïc Crombez

FINDING DIGITAL CONVEXITY

By

LOÏC CROMBEZ

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF COMPUTER SCIENCE

UNIVERSITÉ CLERMONT AUVERGNE
LIMOS
ED SPI

Committe:
Devillers Olivier, Chairman
Coeurjolly David, Committe member
Fekete Sándor, Committe member
Feschet Fabien, Committe member
Silveira Rodrigo, Committe member
Sivignon Isabelle, Committe member
Guilherme Dias Da Fonseca, Advisor
MAY 27, 2021

ACKNOWLEDGMENT

This work has been sponsored by the French government research program "Investissements d'Avenir" through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25).

TABLE OF CONTENTS

LIST OF FIGURES

Dedication

Chapter One

Introduction

Computational geometry studies algorithms that solve geometric problems, and, as a consequence, problems that can be stated in terms of geometry. Some of its applications are related to computer graphics, robotics, computer-aided design, machine learning or augmented reality. The earliest studies related to computational geometry date back to the second half of the 19^{th} century with work on quadratic forms by Dirichlet [26] and Sylvester [58]. Computation geometry emerged in the 1970's from fields such as mathematical programming [14] and computer-aided design [33]. Convex hulls were one of the first topics of attention in computational geometry with a first paper by Chand and Kapur in 1970 [14], and then in 1972 by Graham [36]. Convexity is still to this day very present in computational geometry and will be the main topic of attention of this dissertation.

Digital geometry, a field closely related to computational geometry, studies the geometry of points with integer coordinates, those points are also known as *lattice points* [46]. The motivation behind digital geometry comes from the fact that digital devices such as cameras, lidars or scanners in medical imaging

provide a discrete representation of the real world. As a consequence, continuous geometry is not adapted to work on such inputs and usually involves a transformation on the inputs that implies approximations. In this context, the goal of digital geometry is to build an alternative geometry based on a restricted class of objects such as pixels, voxels, or lattice sets, and relying heavily on integer arithmetic. As a consequence, digital geometry arises a lot of questions concerning the connection that exists between the discrete and the continuous. Despite studies related to digital geometry existing since the end of the 19^{th} century with works such as Minkowski's on the geometry of numbers [52], or results like Pick's theorem [53], digital geometry started in the 1960's for computer graphics, because a screen is just a lattice or a grid of pixels, with algorithms such as Bresenham's [10] line drawing algorithm. From there, questions arose and the field expanded with topics such as digital topology, digital manifolds, the study of properties of digital sets or tomography. Nowadays, some of the main application areas of digital geometry are in image analysis [47] (notably medical imaging [31]), computer graphics [45], and integer linear programming [56].

Not unlike in computational geometry, convexity is a core component in digital geometry. Some of the earliest work related to digital geometry where about counting points in convex polyhedra. Minkowski's theorem [52] relates the number of lattice points inside a convex centrally symmetric polytope to its volume. In 2 dimension, Pick's theorem [53] gives an equality relating the number of lattice points inside and on the boundary of a convex polygon P, whose vertices are lattice points, to the area of P. Studies on this topic of

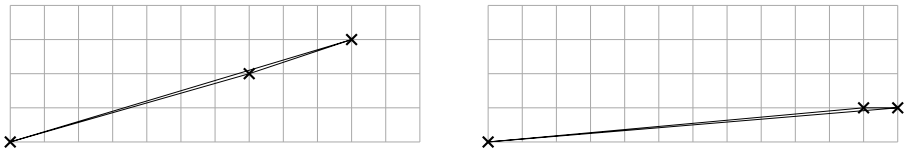

Figure 1.1 Digital Convex Sets. Example of two digital convex sets that are not connected in term of induce grid subgraph.

reporting lattice points inside convex polytopes continued throughout the 20^{th} century and are still considered to this day, with work related to Ehrart's polynomials [29, 49] and Barvinok's algorithm [6]. A natural definition of *digital convexity* that arises is the following:

A lattice set $S \in \mathbb{Z}^d$ is said to be *digital convex* if there exists a convex polytope $P \in \mathbb{R}^d$ such that $P \cap \mathbb{Z}^d = S$. We can easily notice that if such a polytope P exists then the convex hull of S, that we denote $\text{conv}(S)$, also satisfies the property $\text{conv}(S) \cap \mathbb{Z}^d = S$. Hence an equivalent but simpler definition of digital convexity is the following:

A lattice set $S \in \mathbb{Z}^d$ is said to be *digital convex* if $\text{conv}(S) \cap \mathbb{Z}^d = S$. This definition of digital convexity is the one that we will use throughout this dissertation. However, it is worth noting that several other definitions of digital convexity have been investigated throughout the years [16, 18, 41, 42, 44, 48]. These definitions were created in order to guarantee that a digital convex set is connected (in terms of the induced grid subgraph), which simplifies several algorithmic problems. Indeed, in a digital convex set S of n points, the distance between a point p and its nearest neighbour $p_1 \in S$ is not bounded by any function of n. Moreover, The diameter of a digital convex set of at least 2 points can be arbitrarily large (Fig. 1.1).

The presence of convex sets in the core of fundamental problems, as well as their peculiar geometric properties motivates the study of their properties.

1.1 Results

The main results of this dissertation are algorithms related to the recognition of digital convex sets.

First, for any unordered lattice set S of n points, we present an algorithm relying on the quickhull algorithm that tests digital convexity in linear time relative to n. Previous algorithms testing digital convexity in linear time, such as those presented in [11, 25] relied on a specific input representation which meant they couldn't be applied on arbitrary unordered sets. We then use this result to solve the optimal digital convex polygon problem in linear time relative to n. This problem whose decidability has been shown in 2017 [34] asks, for a digital convex set S, to find a polygon $P \in \mathbb{R}^2$ that verifies $P \cap \mathbb{Z}^2 = S$ such that P is minimal in regard to the number of vertices.

Finally, this dissertation considers a problem stated in 2005 in [17], which is the digital versions of the potato peeling problem [15], and presents a polynomial time algorithm that finds, for any lattice set S the largest digital convex subset $C \in S$, as well as the first polynomial time algorithm that finds, for any lattice set S, and any integer $k > 1$ the largest subset $C \in S$ defined as $C = C_1 \cup C_2 \cup ... \cup C_k \in S$ such that for all $i \in 1..k$ C_i is digital convex.

This dissertation is divided as follows.

In Section 2.1, we present the convexity and several convex hull algorithms

introducing classic strategies used in computational geometry, such as "divide and conquer" and angular sorting. One of the presented algorithms will be used in the result presented in this dissertation.

In Section 2.2, we cover some basic concept associated to digital geometry, number geometry and digital convexity.

Section 2.3 presents previous work on simplex range searching, with an emphasis on linear size data structures, as well as data structures that allows logarithmic query time.

In Chapter 3 we present our results on digital convex set recognition, and in Chapter **??** we do the same with our results concerning the digital potato peeling problem.

Finally, in Chapter 5 we present conclusions along with directions for future works in computational digital geometry.

Chapter Two

Literature review

In this chapter, we present terms and results that will be used throughout this dissertation, as well as some other related well known results.

In Section 2.1, we present convexity and its uses in computational geometry, along with several convex hull algorithms. In Section 2.2, we introduce digital geometry and digital convexity. In Section 2.3, we present several simplex range searching structures that are commonly used in computational geometry.

2.1 Convexity

In this section we present the concepts of convexity and convex hull, and we present several known algorithms that compute the convex hull of any set of points. In geometry, a subset S of an Euclidean space is *convex* if and only if for any pair of points $p_1, p_2 \in S$, the line segment $p_1p_2 \in S$. The *convex hull* of a set S, denoted $\text{conv}(S)$ throughout this dissertation, is the smallest convex set such that $S \in \text{conv}(S)$.

2.1.1 The QuickHull Algorithm

Quickhull is one of the many early algorithms to compute the convex hull of a given set of n points S. In dimension 2 its worst-case time is $O(n^2)$. Despite its running time being vastly dependant on the number of vertices on the convex hull, quickhull has shown to be an effective algorithm for inputs having a low number of convex hull vertices relative to the number of input points, such as uniformly distributed input points for instance. Furthermore, for some inputs and variations of the algorithm, the complexity is reduced to $O(n)$ [8, 37].

In order to compute the convex hull of a given set of points S, the quickhull algorithm starts by initializing a convex polygon in the following manner. First it computes the top-most and bottom-most points of the set. Surely, these two points belong to the convex hull. Let ℓ be the line defined by these two points. Then, the algorithm computes the farthest point from ℓ, on both sides of ℓ. The (at most) four points we computed describe a convex polygon that we call a *partial hull*, which is a subset of the vertices of the convex hull of S. All points contained in the interior of the partial hull are discarded from S as they surely do not belong to its convex hull (Fig. 2.1).

After the initialization step, the algorithm adds vertices one by one to the partial hull until it obtains the entire convex hull. For each edge e of the partial hull, the following elimination procedure is applied. Let v denote e's outwards normal vector. The algorithm searches for the extreme point in direction v. If this point's distance from the edge e is 0, then the edge e is an edge of the convex hull. Otherwise, we add to the convex hull the farthest point found, and the points that are inside the new partial hull are discarded. (Fig. 2.2).

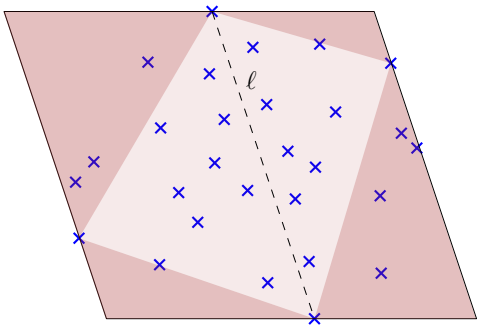

Figure 2.1 Quickhull initialization. Points inside the partial hull (light brown) are discarded. The remaining points are potentially part of the hull.

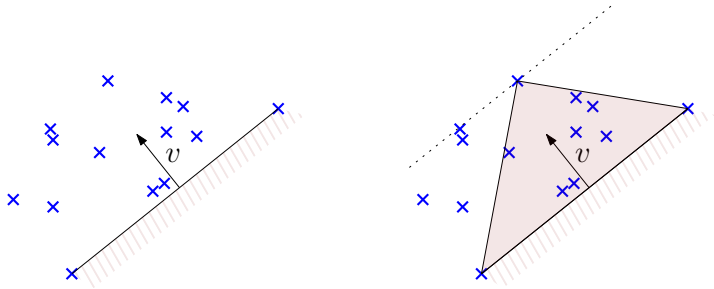

Figure 2.2 Elimination step Points inside the triangle added to the partial hull (light brown) are discarded. The remaining points are potentially part of the hull.

2.1.2 Graham Scan Algorithm

Graham scan [36] is a convex hull algorithm. For an input set S of n points that Graham scan runs in $O(n \log n)$ time, which is the lower bound [60] for convex hull algorithms in the decision tree model. Graham scan relies on the fact that computing the convex hull of a simple polygon can easily be done in linear time relative to the number of vertices of the polygon. Obtaining a simple polygon going through each vertex of a given point set S can be done

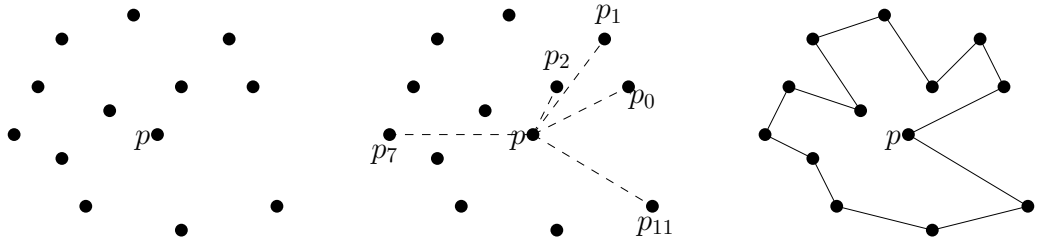

Figure 2.3 Simple polygon from sorting. By sorting all the points around p, we can obtain a simple polygonization of the set S.

picking any point p in S, and sorting the points of S angularly around p. The polygon that visits the points of S in the sorted order is a simple polygon (Fig. 2.3).

Now, assuming the points are sorted clockwise, around p, in order to obtain the convex hull of S we just have to scan through the vertices of the polygon, and remove the points where the polygon makes a right turn. Note that when removing a point from the polygon, the previous vertex has to be tested again for a right turn. This occurs at most n times as any point from the set can only be removed once. The algorithm hence requires $O(n \log n)$ time in order to sort the points around p, and then an additional $O(n)$ time in order to compute the convex hull from the simple polygon. Hence the total running time of the Graham scan algorithm is $O(n \log n)$.

2.1.3 Jarvis March Algorithm

Jarvis March, also known as gift wrapping is an output sensitive convex hull algorithm. In two dimensions, its running time complexity for an input set S of n points is $O(nh)$, where h is the number of vertices on the convex hull. As

a consequence, its worst-case time is $O(n^2)$ when all the points are in a convex position.

Jarvis March starts by computing a point $p_0 \in S$ that is on the convex hull of S, such as the left-most point of S for instance. We then, greedily compute one by one the vertices p_i of the convex hull. Knowing p_{i-1}, and in order to compute p_i, the next point of the convex hull, we simply look for the point p_i such that all points of S are located to the right of the line $p_{i-1}p_i$. We then repeat this process until we reach back to p_0. As computing each p_i takes $O(n)$ time, the total running time of the algorithm is equal to $O(nh)$.

2.1.4 Chan's Algorithm

Chan's algorithm [12] is an output sensitive algorithm that runs in $O(n \log h)$ time in two or three dimensions. It is not the only, nor the first, algorithm that obtains this complexity [43], but it is arguably the simplest. Chan's algorithm is a combination of the Graham scan algorithm 2.1.2, and Jarvis march 2.1.3.

In a first time we describe the algorithm assuming that h, the number of vertices of the convex hull of S, is known. We will then explain how to adapt the algorithm when h is unknown.

First, we start by arbitrarily separating S into at most $1 + \frac{n}{h}$ subsets S_i of at most h points. Then, we compute the convex hull of each S_i using Graham scan. This takes $O(h \log h)$ time for each S_i, as there are at most $1 + \frac{n}{h}$, computing all those convex hulls takes $O(n \log h)$ time.

Second, we use a method similar to Jarvis March to compute the convex hull of S. Hence, knowing p_i a vertex of the convex hull of S, we are looking

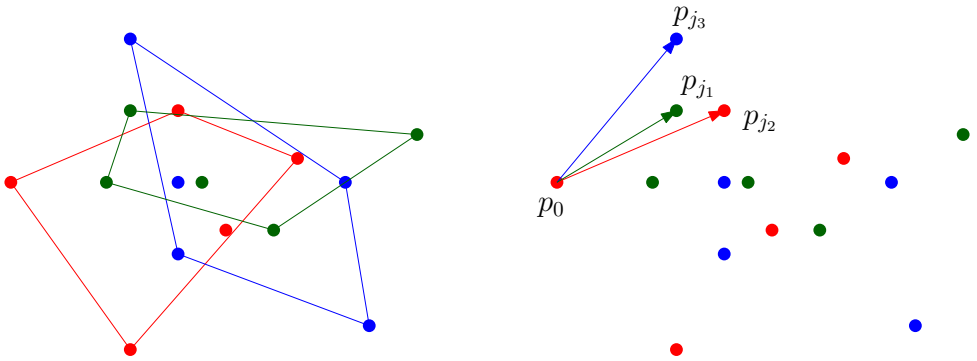

Figure 2.4 Jarvis march step. We are looking for the point p_j such that all the other vertices of the convex hull are located to the right of p_ip_j. Once we computed such a p_{j_i} for each subset S_i, we know that p_j is one of the $_{j_i}$, and hence only to look among those.

for $p_{i+1} \in S$ such that $\forall p_j \in S, j \neq i$ p_j is located to the right of the line p_ip_{i+1}. As p_{i+1} is necessarily in the convex hull of a subset S_i we only have to consider the points that are vertices of those convex hulls. For a given subset S_i, that has at most h points, the points $p_k \in S_i$ such that all $p_j \in S_i$ are located to the right of the line p_ip_k can be computed in $O(\log h)$ using a binary search on the convex hull of S_i (Fig. 2.5). Once we computed the candidate point for each subset, the usual Jarvis march step can be used on the $O(\frac{n}{h})$ candidate points to find p_{i+1} (Fig. 2.4). Hence, overall it takes $O(\frac{n}{h} \log h)$ time to compute p_{i+1}. As we repeat this process exactly h time in order to compute the convex hull of S, the total running time of the algorithm is $O(n \log h)$ when h is known.

We now, explain how we can use the previously described algorithm, that assumes that h is known, in order to compute the convex hull of S in $O(n \log h)$ without knowing h. The idea, is to guess a value for h, and square our guess until we finally reached a value bigger than h. First, we chose a small value,

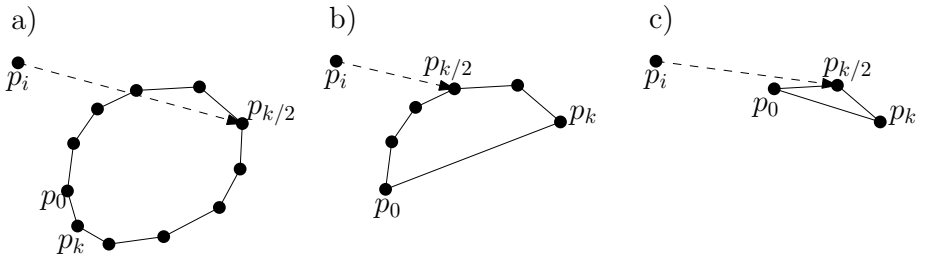

Figure 2.5 Binary search. We are looking for the point p_j such that all the other vertices of the convex hull are located to the right of $p_i p_j$. a) $p_{k/2-1}$ is on the left of $p_i p_{k/2}$, hence p_j is in between p_0 and $p_{k/2}$. b) $p_{k/2+1}$ is on the left of $p_i p_{k/2}$, hence p_j is in between $p_{k/2}$ and p_k. c) Both $p_{k/2-1}$ and $p_{k/2+1}$ are on the right of $p_i p_{k/2}$, hence $p_j = p_{k/2}$.

$h_1 = 2$ for instance. We then run the previously described algorithm until we reach the 3^{rd} step of the algorithm. At this point, the Jarvis march steps either successfully came back to the starting point, which means we found the convex hull, or we realize that h is actually bigger than $h_1 = 3$. In this second situation, we stop computation, and start all over again with $h_2 = h_1^2$. We repeat this process until we guess a value h_k larger than h, which results in us computing the convex hull. For each guess we made, the running time is equal to $O(n \log h_i)$, which result in a total running time of $\sum_{k=0}^{\log \log h} O(n \log 2^{2^k}) = O(n \log h)$.

2.2 Digital Geometry

Digital geometry is the field of mathematics that studies the geometry of points with integer coordinates, also known as *lattice points* [46]. In this section we present some common tools and notions of digital geometry that will be used

throughout this dissertation.

Transformation of $\mathbb{Z}^2$

In most cases, in computational geometry, the points are assumed to be in general position. This is clearly not the case in digital geometry where multiple points often have the same x or y coordinate. Not all affine transformations are bijective from $\mathbb{Z}^2$ to $\mathbb{Z}^2$, but those who do are interesting in the context of digital geometry. We say that those transformations preserve the lattice grid. A transformation matrix that preserves the lattice grid must have integer coefficients, and their determinant must be equal to ± 1. These transformations are called unimodular affine transformations $SL_2(\mathbb{Z})$ [5, 38] when the determinant is equal to one.

One of those transformations is called *shearing* or *transvection* [39]. *Horizontal shearings* are the linear transformations whose transformations matrices are of the form $\begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix}$, $k \in \mathbb{Z}^*$. Similarly, we call *vertical shearing* a linear transformation whose transformation matrix is $\begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix}$, $k \in \mathbb{Z}^*$. It is clear that both horizontal and vertical shearing define a bijection from $\mathbb{Z}^2$ to $\mathbb{Z}^2$ (Fig. 2.6).

It is interesting to note that shearings generate $SL_2(\mathbb{Z})$ [3]. For a given linear transformation $\mathcal{L} \in SL_2(\mathbb{Z})$, as the determinant of the matrix representing $\mathcal{L}$ is equal to 1, the composition of shearings equivalent to $\mathcal{L}$ can be obtained simply by using the extended Euclidean algorithm. Lemma 1 states that for any line ℓ going through any two lattice points, there is a linear transformation

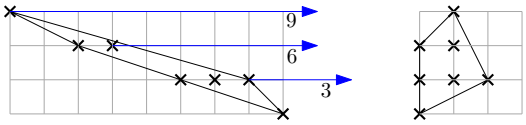

Figure 2.6 Shearing. On the left: A set of points before a horizontal shearing of three. Relatively to the bottom row, points from the second row are moved to the right by three, points from the third row are moved to the right by six, and so on. On the right: The same set of points after the shearing.

$\mathcal{L} \in SL_2(\mathbb{Z})$ that maps ℓ to a horizontal line. Lines supported by at least two lattice points are called *Diophantine lines* and actually go through an infinite number of lattice points. Their equations are of the form $ax + by = c$, such that a,b, and c are integers, and such that a and b are co-prime.

Lemma 1. *For every Diophantine line ℓ there exists a linear map $\mathcal{L} \in SL_2(\mathbb{Z})$ such that $Im_{\mathcal{L}}(\ell)$ is an horizontal line.*

Proof. Let ℓ be a Diophantine line. Let $p(x_p, y_p)$ and $q(x_p + a, y_p + b)$ be two consecutive lattice points on ℓ, that is the line segment pq only contains two lattice points, namely p and q. This implies that we have $\mathrm{GCD}(a, b) = 1$. Let x and y be the Bézout coefficients of a and b, those are the two integers x and y such that $ax + by = 1$. We now consider the matrix $M = \begin{bmatrix} x & y \\ -b & a \end{bmatrix}$. By construction $det(M) = ax + by = 1$. Hence M defines a linear map $\mathcal{L}$ such that

$$Im_{\mathcal{L}}(p) = Mp = \begin{bmatrix} xx_p - yy_p \\ -bx_p + ay_p \end{bmatrix}$$

$$Im_{\mathcal{L}}(q) = \begin{bmatrix} xx_p + ax - yy_p + by \\ -bx_p - ab + ay_p + ab \end{bmatrix} = \begin{bmatrix} xx_p - yy_p + 1 \\ -bx_p + ay_p \end{bmatrix} = Im_{\mathcal{L}}(p) + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Hence $Im_{\mathcal{L}}(d)$ is an horizontal line and $Im_{\mathcal{L}}(\mathbb{Z}^2) = \mathbb{Z}^2$. □

We say that two Diophantine lines $\ell_1 : a_1x+b_1y = c_1$ and $\ell_2 : a_2x+b_2y = c_2$ are *consecutive* if $a_1 = a_2$, $b_1 = b_2$ and $|c_1 - c_2| = 1$. Lemma 2 states that no lattice point is located in between ℓ_1 and ℓ_2, and a corollary of this lemma is that for any co-primes a and b, all points from $\mathbb{Z}^2$ are located on a Diophantine line of equation $ax + by = k$, $k \in \mathbb{Z}$.

Lemma 2. *For any pair a and b of co-prime numbers, for any two consecutive Diophantine lines $\ell_1 : ax + by = c$ and $\ell_2 : ax + by = c + 1$. There are no lattice points located in between ℓ_1 and ℓ_2.*

Proof. Consider a lattice point $p(x_p, y_p)$, as a, b, x_p, and y_p are integers so is $ax_p + by_p$, which hence cannot be in the interval $(c, c + 1)$ □

Digital Connectivity

Connectivity, does not have the same meaning in digital geometry as in classic Euclidean geometry. However, two simple notions of connectivity are used in digital geometry in the plane. We say that two lattice points p_1 and p_2 are *8-connected* (resp. *4-connected*) if their Chebyshev distance L_∞ (resp. Euclidean distance) is one. We say that a set S of lattice points is *8-connected* (resp. *4-connected*) if for all pair of points p_1 and p_2 in S there is a chain of *8-connected* (resp. *4-connected*) points in S going from p_1 to p_2.

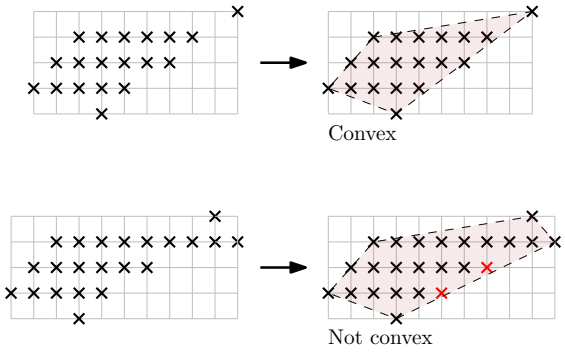

Figure 2.7 Digital convexity. The first set is digital convex, while the second set is not because of the red lattice points that are inside the convex hull of the set but not in the set itself.

Digital Convexity

Although the subsets of $\mathbb{Z}^d$ are not convex in the usual meaning of the term, a simple notion of convexity is induced by the convexity of $\mathbb{R}^d$ [55]. A set of lattice points $S \subset \mathbb{Z}^d$ is *digital convex* if $\text{conv}(S) \cap \mathbb{Z}^d = S$, where $\text{conv}(S)$ denotes the convex hull of S in $\mathbb{R}^d$ (Fig. 2.7). In other words, S is digital convex if it is the intersection of a convex subset of $\mathbb{R}^d$ with the lattice $\mathbb{Z}^d$. Digital convex lattice sets are then directly related to the lattice polytopes investigated in geometry of numbers since the works of Minkowski [52]. Digital convexity is preserved by homeomorphisms of $\mathbb{Z}^d$.

Let us remark that a digital convex set S is not necessarily connected while the convex sets of $\mathbb{R}^d$ are arc-connected or simply connected. In $\mathbb{Z}^2$ and $\mathbb{Z}^3$, the lack of connectivity has, throughout the years and today still, motivated the introduction of some alternative definitions of digital convexity [16, 18, 41, 42, 44, 48] that we will not consider here. Despite this lack of connectivity, different bounds have been established between the different characteristics of

a digital convex set S. We denote n be the number of lattice points in any lattice set S, h the number of edges of $\text{conv}(S)$, and r the diameter (largest Euclidean distance between two points) of S. It is clear that for a given number of edges h, the number of points n and the diameter r are not bounded (consider for instance very long and skinny triangles). Similarly, for a given number of points n, the diameter r is not bounded (consider the pair of points $(0,0)$ and $(1,r)$). However the number of vertices (or edges) h is bounded by $O(n^{1/3})$ [23]. At last, given the diameter r, the number of points n is clearly at most $O(r^2)$ and h is at most $O(r^{2/3})$ [61].

Some other measures, specific to digital geometry, also exists. For instance, the lattice diameter $\ell(S)$ of a digital convex set S is the measure of the longest string of co-aligned lattice points in S. The width also has its digital analogous, and the lattice width is defined as follow in digital geometry. We first define $\omega_u(S)$ the width of S along a direction $u \neq (0,0)$ as:

$$\omega_u(S) = \max_{x,y \in S}(u(x-y))$$

More intuitively, the width of S along a direction u is equal to -1 plus the number of consecutive parallel line supported by lattice points and of direction orthogonal to u that are required to cover S (Fig. 2.8). The lattice width of S is:

$$\omega(S) = \min_{u \in \mathbb{Z}^2 \setminus (0,0)} \omega_u(S)$$

It is worth noting that both lattice diameter and lattice width are invariant under the group of unimodular affine transformations, and that $\omega(S) \leq \frac{4}{3}\ell(S) + 1$ [4].

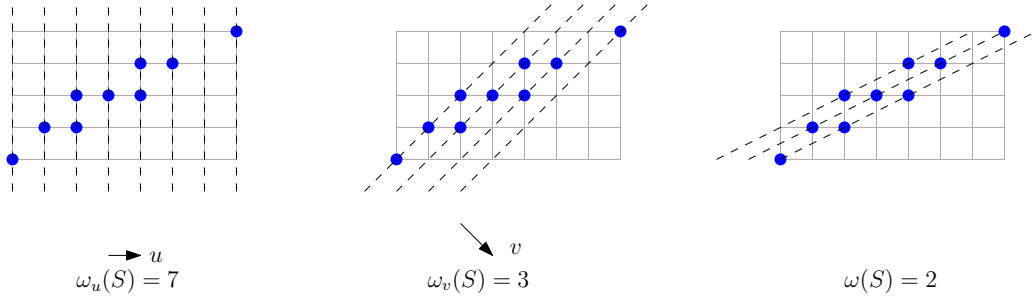

Figure 2.8 Lattice width. Example of width of a lattice set along two directions, and its lattice width.

2.3 Range Searching

In this section we present existing works on range searching. Section 2.3.1 describes a triangle range counting algorithm that we will make use of in Section 4.1, whereas Section 2.3 proposes a quick overview of existing work on simplex range searching.

2.3.1 Triangular Range Counting

In this section, we present a preprocessing method in $O(n^2)$ time and space, that for a set of points S allows to retrieve the number of points inside any triangle in S in constant time. Note that the triangles requested must have their vertices in S, which is why the query time is better than the known lower bound on generic polytope range searching [20]. The structure introduced in [30] stores, for each pair of points (p_1, p_2) in S the number of points in the vertical strip below the line segment p_1p_2. Let $below(ab)$ denote the number of points below the line segment ab. The request can compute the number of points inside any triangle p, q, r (p the left-most point and r the right-most

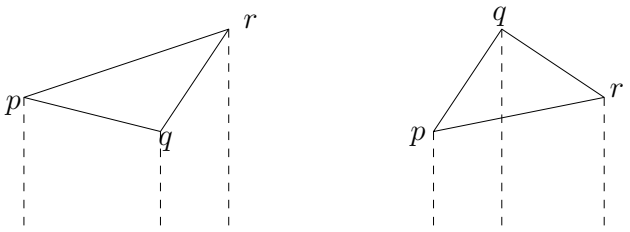

Figure 2.9 Range counting. The two possibilities for the triangle pqr. On the left: we have to remove the points below pq and qr from the points below pr. On the right: we have to remove the points below pr from the points below pq and qr.

one) thanks to the fact that the number of points inside the triangle is equal to $|below(pq) + below(qr) - below(pr)|$ (Fig. 2.9) which takes $O(1)$ time given the preprocessing.

In order to compute the number of points below each segment, the line segments are treated from left to right according to the right-most point. All line segments with the same right-most point r are treated in clockwise order in the following manner:

- All points located to the left of r are sorted in clockwise order around r $(p_1, p_2, ..., p_k)$.

- For all p_i in order, we compute $below(p_i r)$ in the following manner:

  If p_i lies to the left of p_{i-1} then

  $below(p_i r) = below(p_{i-1} r) + below(p_i p_{i-1}) + 1$.

  Else p_i lies to the right of p_{i-1} then

  $below(p_i r) = below(p_{i-1} r) - below(p_{i-1} p_i)$.

  As p_{i-1} and p_i are consecutive in the clockwise ordering around p, we know that the triangle $p_{i-1} p_i r$ does not contain any other point from S,

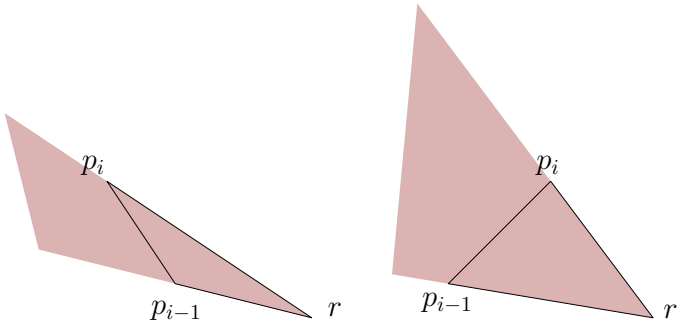

Figure 2.10 Range counting preprocessing. In both cases the dark red cones contain no points from S. On the left: The points below $p_i r$ are the points below $p_i p_{i-1}$ and the points below $p_{i-1} r$. One the right: The points below $p_i r$ are the points below $p_{i-1} r$ minus the points below $p_{i-1} p_i$.

and hence the method we just describe correctly computes the number of points below the edges whose right-most point is r (Fig.2.10).

Sorting all points clockwise around each point can be done in $O(n^2)$ time using the dual [8, Chapter 11]. The remaining of the computation takes $O(n)$ time for each of the $O(n)$ right-most points, hence the total preprocessing time is $O(n^2)$.

2.3.2 Simplex Range Searching

In this section, we propose an overview of the history of data structures used to solve simplex range searching problems. We will first look at data structures that allow a logarithmic query time, and then will consider the data structures that only use linear space. More often than not, the data structures used in order to solve simplex range searching problem are partition trees. A partition tree T is a data structure that partitions a point set S into subsets. Each point

of S is stored in exactly one leaf of T, and each leaf contains at most a constant number of points. Each node N of T stores a polyhedron $P(N)$ of size $O(1)$. All points stored in the leaves underneath N are enclosed by $P(N)$, and no other point in S is enclose by $P(N)$. In addition, each node N stores the number of points enclosed by $P(N)$.

Given a set H of n hyperplanes $H_1, H_2, ..., H_n$, an *ϵ-cutting* $\mathcal{C}$ is a division of $\mathcal{R}^d$ into simplices $C_0, C_1, ...C_k$ such that the C_i are mutually disjoint and the interior of any C_i is intersected by at most ϵn hyperplanes H_i.

Linear size data structures

The majority of linear size data structures used to solve simplex range searching problems are based on *partition trees*. Given a set S of n points in $\mathbb{R}^d$, a *partition tree* partitions the space into a small number of regions, each containing approximately the same number of points, then recursively partition each region in a similar way. Partition trees where first introduced in the plane in [59] and relied on the following *ham-sandwich theorem* [28].

Theorem 3. *For any two sets S_1 and S_2 of n_1 and n_2 points in the plane, there is a halfplane h such that $S_1 \cap h = \left\lfloor \frac{n_1}{2} \right\rfloor$ and $S_2 \cap h = \left\lfloor \frac{n_2}{2} \right\rfloor$.*

Using the *ham-sandwich theorem*, we find two hyperplanes h_1 and h_2 such that each one of the four quadrants induced by h_1 and h_2 contains $\frac{n}{4}$ points. The root of the partition tree stores h_1, h_2 and n. Then, for each quadrant, we recursively construct a subtree in the same manner. The total size of the data structure is proportional to $\sum_{k=0}^{\log_4 n} 4^k$ and is hence linear, and can be computed in $O(n \log n)$ time. Using this partition tree, a halfplane h range

counting query can be answered as follows. For each of the four quadrants Q_i attached to the root of the tree we do the following. If the line l_h induced by the halfplane h intersects the quadrant, then we visit the children of Q_i. If $Q_i \cap h = \emptyset$, then we do nothing. Finally, if $Q_i \subset h$, we add all the points in Q_i to the global count. As the quadrant are induced by two lines, l_h intersects at most three quadrants, the query time is $O(n^{\log_4 3})$. The same procedure can be applied to answer simplex range counting queries in the same time complexity, and simplex range reporting in $O(n^{\log_4 3} + k)$, where k is the number of points reported.

A data structure that reaches an optimal worst case query time of $O(n^{1-1/d})$, where d is the dimension, for linear structure [21] in the arithmetic model was presented in [51]. This data structure is based on the following partition tree theorem from [50]

Theorem 4. *For any set S of n points in $\mathbb{R}^d$, and any r such that $1 < r < \frac{n}{2}$ there is a family of pairs set/simplex $\Gamma = (S_1, P_1), ..., (S_i, P_i), ..., (S_k, P_k)$ such that for each i, $S_i \subset S$ is located inside P_i, for all $i \neq j$ $S_i \cap S_j = \emptyset$, and $\frac{n}{r} \leq |S_i| \neq \frac{2n}{r}$, and there is a constant c such that any hyperplane crosses at most $cr^{1-\frac{1}{d}}$ P_i.*

A partition tree can be built by computing the partition described in the previous theorem 4. For each simplex P_i, we recursively construct a subtree in the same manner. The total size of the data structure is linear, and can be computed in $O(n \log n)$ time. Using this partition tree, a halfspace h range counting query can be answered as follows. For each simplex P_i attached to the root of the tree we do the following. If the hyperplane l_h that is the boundary

of the halfplane h intersects P_i, then we visit the children of P_i. If $P_i \cap h = \emptyset$, then we do nothing. Finally, if $P_i \subset h$, we add all the points in P_i to the global count. As l_h crosses at most $cr^{1-\frac{1}{d}}$ simplex, the query time is $O(n^{1-\frac{1}{d}+\log_r c})$, by choosing $r = n^\epsilon$, this results to a query time of $O(n^{1-\frac{1}{d}} \log^{O(1)} n)$.

Finally, a data structure with $O(n^{1-1/d} \log n)$ query time, $O(n)$ space, and $O(n \log n)$ preprocessing was presented by Chan [13], and is based on the two following results.

Lemma 5. *For any set S of n points in $\mathbb{R}^d$, there is a set H of $n^{O(1)}$ hyperplanes such that for any collection of disjoint cells each containing at least one point $p \in S$, if k is the maximum number of cells crossed by a hyperplane in H, then the maximum number of cells crossed by any hyperplane is $O(k)$.*

Theorem 6. *For any set S of n points, let H be a set of m hyperplanes in $\mathbb{R}^d$. Given l disjoint cells covering S such that each cell contains at most $2n/l$ points of S and each hyperplane in H crosses at most k cells. Then, for any constant c, every cell can be subdivided into $O(c)$ disjoint subcells such that each subcell contains at most $2n/(cl)$ points in S, and each hyperplane crosses at most $O((cl)^{1-1/d} + c^{1-1/(d-1)}k + c \log l \log m)$*

Using the set H of hyperplanes from Lemma 5, and successively applying Theorem 6 to the tree consisting of simply a root cell containing all n points of S, we successively construct hierarchies denoted $\Pi_1, \Pi_c, \Pi_{C^2} ...$ of $1, c, c^2, ...$ cells. Π_c gives the first depth of a partition tree, then Π_{c^2} results in the second depth and so on. In the end, we obtain a partition tree of degree $O(c)$. Let $k(c^j)$ denote the maximum number of cells of Π_{c^j} crossed by any hyperplane,

then: $k(c^{j+1}) \leq O((cl)^{1-1/d} + c^{1-1/(d-1)}k(c^j) + c \log l \log n)$. The resulting tree has height $O(\log n)$, order $(1 - 1/d)$ and query cost $O(n^{1-1/d} \log n)$.

Logarithmic query time data structures

In order to solve simplex range searching, we will first consider the simpler halfspace range counting problem, in which we only want to retrieve the number points in a given halfspace. Using duality, and the following property of duality that states that a point p is above a hyperplane h if and only if the dual point $h*$ of h is above the dual hyperplane $p*$ of p, we can see that the halfspace range counting problem is actually equivalent to counting the number of halfplanes that are located above a query point.

To do so, we make use of the following cutting theorem presented in [19]:

Theorem 7. *For any set of n hyperplanes H in $\mathbb{R}^d$, $r \leq n$, and constant $b > 1$, there exist $k = \log_b r$ cuttings $C_1, C_2, ..., C_k$ such that C_i is a $\frac{1}{b^i}$ cutting of size $O(b^{id})$. Each C_i is composed of simplices each contained in a simplex of C_{i-1}, and each simplex C_i contains a constant number of simplices of C_{i+1}. Such a cutting can be computed in $O(nr^{d-1})$ time.*

By choosing $r = \frac{n}{\log_2 n}$, and constructing the cuttings from theorem 7 Chazelle [19] achieved a data structures of size $O(\frac{n^d}{\log^{d-1} n})$ that can answer a halfspace range query in $O(\log n)$ time in the following manner, using the dual. For each simplex S of C_i, we store the simplices of C_{i+1} that are located in C_i, and the number of hyperplanes located above S. For the last layer of the cutting, C_k, we also have to store the list of hyperplanes that intersects each simplex.

In order to answer a query, we simply go through the tree in order to find the simplex $S \in C_k$ that contains our query point p. We then simply return the number of hyperplanes above S plus the number of hyperplanes intersecting S above p.

Chapter Three

Digital Convex Set Recognition

In this chapter, we develop algorithms to recognize digital convex sets.

In Section 3.1, we present a near linear time algorithm to test digital convexity. The algorithm is based on the quickhull algorithm presented in Section 2.1.1.

In Section 3.2, we present a near linear time algorithm that given any digital convex set S finds a polygon P with vertices in $\mathbb{R}^2$ with minimum number of edges such that $P \cap \mathbb{Z}^2 = S$.

3.1 Testing Digital Convexity

In this section, we consider the question of determining whether a given finite lattice set S is digital convex. Previous related works considered structured data in which S is assumed to be connected [11, 25]. Notice also that if the set is ordered, its convex hull can be computed in linear time. In this section we consider the input to be an unstructured lattice sets.

Problem `Test Convexity`

Input: A set $S \subset \mathbb{Z}^2$ of n lattice points given by their coordinates.

Output: Determine whether S is digital convex or not.

Herein we present an algorithm that solves the `Test Convexity` problem in $O(n+h\log r)$ time, where $n = |S|$, h is the number of edges of $\text{conv}(S)$, and r is the diameter of S. The algorithm makes use of the quickhull algorithm and relies on the following Theorem 8 that states that the quickhull algorithm is able to find the convex hull of any digital convex set S in linear time and space relative to the cardinality of S.

Theorem 8. *If the input is a digital convex set of n points, then the quickhull algorithm has $O(n)$ time and space complexities.*

Proof. During the quickhull algorithm, we discard from S the points located inside the partial hull, and add some of them as vertices of the partial hull. Theorem 8 is a consequence of the following two propositions, which we prove next: (i) At least half of the remaining points are discarded at each iteration. (ii) At each step, the running time is linear in the number of points remaining in S. We start by proving proposition (i).

We consider one step of the quickhull algorithm as described in 2.1.1. Let ab be the edge of this step. When the point a was added to the hull, it was the farthest point in a given direction. Hence, there is no point beyond the line orthogonal to this direction going through a. We call this line ℓ_a (Fig. 3.1-b). The same holds for point b, we call the associated line ℓ_b. Let c be the intersection point of ℓ_a and ℓ_b. We know that any remaining point of S

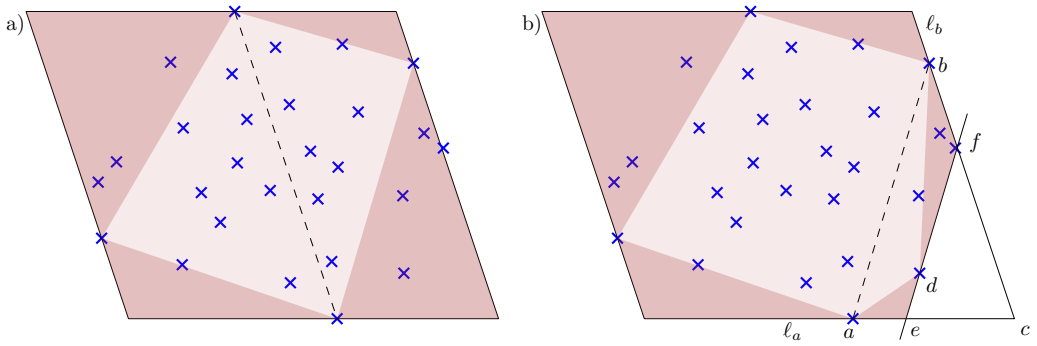

Figure 3.1 Quickhull regions. The preserved region (region in which we look for the next vertex to be added to the partial hull) is a triangle. This stays true when adding new vertices to the hull (as shown here in the bottom right corner). The partial hull (whose interior is shown in light brown) grows at each vertex insertion to the partial hull. The points in or on the boundary of the new region of the partial hull are discarded.

in the outward direction of ab relative to the partial hull is in the interior of $\triangle abc$. We proceed with the remaining points of S in $\triangle abc$ as follows. We are looking for the point that is the farthest from the line ab in the triangle $\triangle abc$ (Fig. 3.2). Three cases might occur. If the triangle $\triangle abc$ does not contain any remaining point, then ab is an edge of the partial hull and we stop the computation for this edge in the following steps. If there is a unique remaining point of the triangle $\triangle abc$ which is the farthest from the line ab, then we denote it d. If there are multiple points which are farthest from ab in the interior of the triangle $\triangle abc$, we denote the two extreme points of S on this segment d and d'.

Let us consider the case where the point d is the unique farthest point from the line ab. Let e and f be the intersections between the line parallel to ab going through d, and respectively ac and bc. The point d is the unique

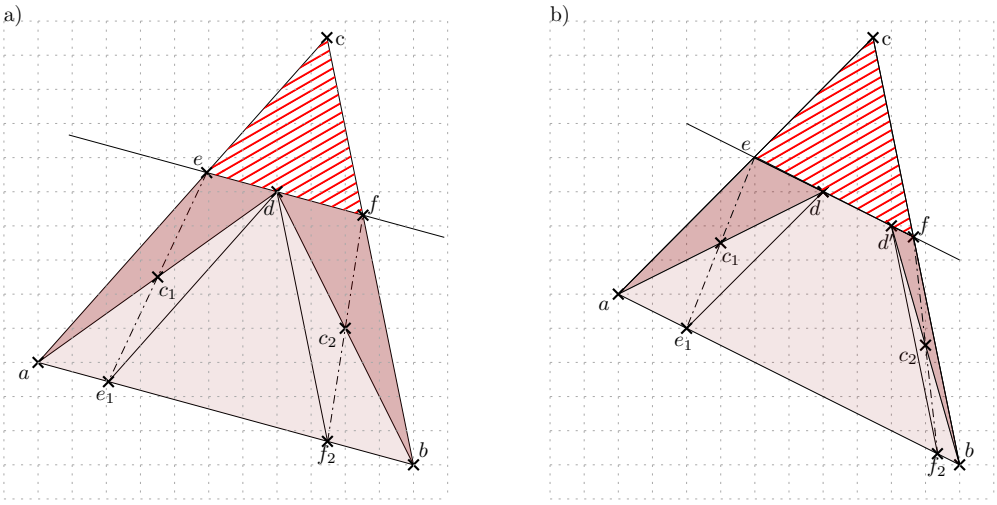

Figure 3.2 Symmetrical regions. At each step, we discard from S all the points of the triangle $\triangle abc$ which are not in the interior of $\triangle ade$ or of $\triangle dbf$ ($\triangle d'bf$ in b). By considering the symmetries through c_1 and c_2, any of these remaining points has a symmetric lattice point in the interior of $\triangle abd$ which is discarded.

remaining point in the triangle $\triangle cef$. (Fig. 3.1-b) Adding d to the partial hull creates two other edges to be further processed: one is ad and the other is bd. Then we insert the vertex d in the partial hull and remove from S all the points which are neither in the interior of the triangles $\triangle ade$ nor $\triangle bdf$. The points of S in the interior of the triangle $\triangle abc$ that we do not discard are allocated either to the interior of $\triangle ade$ or $\triangle bdf$ according to their positions. It is important to note for the complexity of the algorithm that each point is allocated to at most one triangle.

We denote respectively c_1 and c_2 the midpoints of ad and bd. All the lattice points in the interior of the triangles $\triangle ade$ and $\triangle dbf$ have different symmetric lattice points towards c_1 and c_2 in the interior the triangle $\triangle ade$. Since S is digital convex, those lattice points are in S, they also are discarded due to

their positions. (Fig. 3.2-a). In other words, at this step, for each remaining point of S, one point of S is discarded. It proves (i).

This proposition also holds in the case where there are two extreme points d and d' from S on the line ef. In this case, we insert the two vertices d and d' in the partial hull. We discard from S all the points of the triangle $\triangle abc$ which are not in the interiors of the triangles $\triangle ade$ and $\triangle d'bf$. As previously, any of the remaining points has a different symmetric point which is discarded (Fig. 3.2-b). It proves (i) in this case. In both cases our initial assumption is preserved: all the remaining points are in the interior of the triangle to which they are allocated. At last, we can easily provide an initialization of the partial hull and of the set of remaining points satisfying this condition.

For proving (ii), the computation of the farthest point from the line ab among the remaining points of S in the triangle $\triangle abc$ takes linear time. For all points in the triangle we test if they are in the interior of either the triangles $\triangle ade$ or $\triangle dbf$ (or $\triangle d'bf$ in the second case). We allocate them to their containing triangle or discard them. The operation takes constant time per point. In the second case, where we have two extreme points d and d', these two points are also computed in linear time; This proves (ii). Consequently, the number of operations at each step is proportional to the number of remaining points, which is at most half the number of points of the previous step. Therefore the total number of operations is bounded by $n\sum_{i=0}^{\infty}(\frac{1}{2})^i = 2n$, hence the quickhull algorithm runs in linear time for digital convex sets.

□

By running quickhull on any given set S, and stopping the computation if

any step of the algorithm discards less than half of the remaining points, we ensure both that the running time is linear, and that quickhull finishes for any set S that is digital convex. However, if the computation finishes for S, we still need to test its digital convexity. To do so, we use the computed convex hull.

If the number of convex hull vertices h is larger than $(8\pi^2 n)^{1/3}$, then S is not digital convex (see [2, 54], the upper bound $h \leq (8\pi^2 A)^{1/3}$ is given according to the area A of the convex hull of a digital convex set S, but if S is not a set of colinear points, Pick's formula gives $A < n$ which gives $h \leq (8\pi^2 n)^{1/3}$ for convex lattice sets where $n = |S|$). We can assume that h is lower than $(8\pi^2 n)^{1/3}$. Then we compute $|\operatorname{conv}(S) \cap \mathbb{Z}^2|$ using Pick's formula [53]. The set S is digital convex if $|\operatorname{conv}(S) \cap \mathbb{Z}^2| = |S|$.

Theorem 9. *The digital convexity of a set S can be tested in $O(n + h \log r)$ time, where $h = |\operatorname{conv}(S)| \leq O(n^{1/3})$ and r is the diameter of S.*

Proof. As we run the quickhull algorithm, but stop if less than half of the remaining points have been removed, the running time of the quickhull part is bounded by the series $n \sum_{i=0}^{\infty} (\frac{1}{2})^i = 2n$, and is hence linear. If the quickhull algorithm has been stopped, then the set S is not digital convex. Otherwise, if the number of convex hull vertices h is larger than $(8\pi^2 n)^{1/3}$, then we know that the set S is not digital convex. We now consider the remaining case where $h \leq (8\pi^2 n)^{1/3}$. Computing $|\operatorname{conv}(S) \cap \mathbb{Z}^2|$ using Pick's formula requires the computation of both the area of $\operatorname{conv}(S)$ in $O(h)$ time and the number of boundary lattice points, which requires the computation of a greatest common divisor for every edge. Hence, this takes $O(h \log r)$ time where $h = |\operatorname{conv}(S)|$

and r is the diameter of S. As S is digital convex if and only if $|S| = |\operatorname{conv}(S) \cap \mathbb{Z}^2|$, we can effectively test digital convexity in $O(n + h \log r)$ time. □

3.2 Optimal Digital Convex Polygon

In this section, as in Section 3.1, we consider the fundamental question of pattern recognition that is the recognition of digital convex polygons. In this version of the problem, we are given a set $S \subset \mathbb{Z}^2$ of n points and an integer q, the goal is to determine the existence of a convex polygon P with q edges such that $P \cap \mathbb{Z}^2 = S$. Notice that in this problem the vertices of P are not necessarily lattice points. We provide an algorithm to solve the planar recognition of digital convex polygons in linear time. The algorithm is more general and actually solves the following minimization problem:

Problem 1. *Edge Minimization*

Input: *Set $S \subset \mathbb{Z}^2$ of n lattice points given by their coordinates.*

Output: *A convex polygon P with minimum number q of edges verifying $P \cap \mathbb{Z}^2 = S$.*

We note that the problem `Edge minimization` can be rephrased as the following polygonal separation problem with the set $IN = S$ and its complement $OUT = \mathbb{Z}^2 \setminus S$ (Fig. 3.3).

Problem 2. *Polygonal Separation*

Input: *A set $IN \subset \mathbb{Z}^2$ of inliers and a set $OUT \subset \mathbb{Z}^2$ of outliers.*

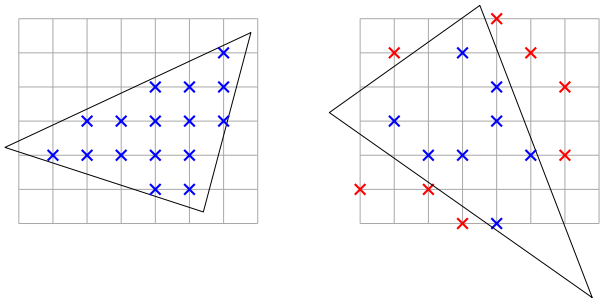

Figure 3.3 Edge Minimization and Polygonal Separation. To the left, an example of the edge minimization problem. The lattice set represented by the blue points is separated from the remainder of $\mathbb{Z}^2$ by the black triangle. To the right, an example of the polygonal separation problem. The black triangle encloses all points from the blue set and none from the red set.

Output: *A convex polygon $P \subset \mathbb{R}^2$ with as few edges as possible and such that all points of IN and none of OUT are inside P.*

Polygonal separability has been widely investigated in the literature. An optimal algorithm for `Polygonal Separation` that takes $O((|IN| + |OUT|) \log(|IN| + |OUT|))$ time is presented in [27]. However, it cannot be directly applied to `Edge minimization` since the set of outliers $OUT = \mathbb{Z}^2 \setminus S$ is not finite.

In order to solve `Edge minimization` we use the following three steps strategy:

The first step consists in testing the digital convexity of S in linear time using Theorem 9. If S is not digital convex, then there is no solution to the `Edge minimization` problem. Otherwise, the quickhull algorithm computes the convex hull of S in $O(n)$ time and we can proceed to the second step.

The second step of the algorithm takes the $h \leq O(n^{1/3})$ edges of $\text{conv}(S)$ as

an input and consists in reducing the set of outliers $OUT = \mathbb{Z}^2 \setminus S$ to a finite subset $OUT' \subseteq OUT$ of $O(n)$ points. The cardinality of OUT' is proportional to the number of lattice points located on the edges of $\mathrm{conv}(S)$, which is $O(n)$ in the worst case. To keep the complexity of the step almost linear in h we do not explicitly compute OUT'. Instead, we compute an implicit description of OUT' of size $O(h)$ in $O(h \log r)$ time, where h is the number of edges of $\mathrm{conv}(S)$.

In the third step we separate OUT' from S using the smallest number of edges possible. To do so we could use the polygonal separability algorithm from [27], but that would lead to a running time of $O(n \log r + n \log n) = O(n \log r)$. Instead, we provide an algorithm that takes benefit of the lattice structure to achieve a running time of $O(h \log r)$ for this step. After the convex hull computation and digital convexity tests of the first step, the whole algorithm takes $O(n+h \log r)$ time. However, if the convex hull of S is provided the algorithm runs in $O(h \log r)$ time.

As the first step, i.e. testing the digital convexity, is already addressed in the previous Section 3.1, we present the second and third steps of the algorithm in the two following sections.

3.2.1 Outliers Reduction

In this section we assume that the set S is digital convex and show how to reduce the set of outliers $OUT = \mathbb{Z}^2 \setminus S$ to a finite set of $O(n)$ points. To do this, we use the notion of jewels introduced in [22, 35] for testing digital circularity and recognizing digital polyhedra. We say that a point $p \in \mathbb{Z}^2 \setminus S$ is

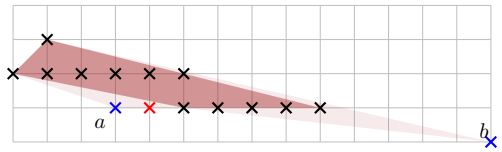

Figure 3.4 Jewel's hull. In black, the set S, its convex hull is in dark red. The point a is not a jewel because of the red point, any convex polygon that includes both S and a also includes the red point. The point b is a jewel because its union $S \cap \{b\}$ with S is still convex. In other words, the convex hull of the union $S \cap \{b\}$ does not contain any other lattice points.

a *jewel* of S if $\text{conv}(S \cup p) \cap \mathbb{Z}^2 = S \cup p$ (Fig. 3.4). The set of all the jewels of S is denoted Jewel(S) and it has the property that a convex set separates S from $\mathbb{Z}^2 \setminus S$ if and only if it separates separates S from Jewel(S) [35]. Hence, the infinite set of the outliers of our separability can be reduced from $OUT = \mathbb{Z}^2 \setminus S$ to $OUT' = \text{Jewel}(S)$.

It has been proven that the number of jewels is infinite if and only if S is the intersection of a line segment and $\mathbb{Z}^2$ [35]. In this case it is clear that the set S forms a digital triangle. A simple way to establish bounds on the number of jewels has been discovered by French high school students during the national contest TFJM2017 https://tfjm.org/editions-precedentes/edition-2017. They presented the following structure of the set of jewels: the jewels of the lattice set S are the lattice points located on the edges of a polygon J surrounding the convex hull of S. This surrounding polygon $J \supset \text{conv}(S)$ is the arithmetic dilation of conv(S) obtained by moving the support lines of the edges of the conv(S) to the next Diophantine lines towards the exterior (Fig. 3.5). We define J as the *jewel hull* of S (Fig. 3.5) and define it more formally as follows.

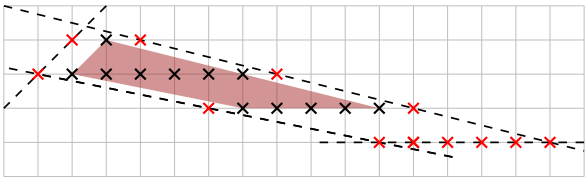

Figure 3.5 Jewels. In black, the set S, its convex hull is in dark red. The halfplanes H'_i are delimited by the dashed lines, and form the *jewel hull* that surrounds the convex hull of S. The jewel hull has three properties: its edges are parallel to the ones of the convex hull of S, there are no point between the convex hull and the jewel hull and all the jewels (drawn in red) are on its boundary.

Given S, let $E = \{e_1, e_2, ...e_h\}$ be the edges of conv(S). For each i from 1 to h, let $HP_i : a_i x + b_i y + c_i \leq 0$ (a_i and b_i co-prime integers) be the closed supporting halfplane associated with the edge e_i such that $S \subset HP_i$. Notice that conv$(S) = \bigcap_i HP_i$. Consider the open halfplanes $HP'_i : a_i x + b_i y + c_i < 1$. Notice that there is no integer point in $HP'_i \setminus HP_i$. The *jewel hull* of S is the closure of the intersection of the half-planes HP'_i (Fig. 3.5).

The jewel hull J of S has three main properties. (i) By construction, its edges are parallel to the edges of conv(S). (ii) It is clear that there is no integer point in the surface located between conv(S) and the jewel hull J. Finally, (iii) A corollary of the next lemma is that the jewels of S are a subset of J, more precisely $jewel(S) = (J \cap \mathbb{Z}^2) \setminus S$.

Lemma 10. *For any three lattice points p_1, p_2, p_3 such that p_1, p_2 are located on the line $ax + by + c = 0$ (coefficients a and b are coprime) and p_3 does not, we have that the triangle $p_1p_2p_3$ either contains a lattice point on the line $ax + by + c + 1 = 0$ or on the line $ax + by + c - 1 = 0$.*

Proof. Up to a lattice preserving affine isomorphism, we can assume $p_1 = (0, 0)$

and $p_2 = (0, u)$ while the images of the two lines are $x = -1$ and $x = 1$. We assume p_3 is located on the right of p_1p_2 (the other case is identical by symmetry). Hence, there exists three integers u, v, w with $u, v > 0$ such that $p_1 = (0, 0)$, $p_2 = (0, u)$, and $p_3 = (v, w)$ and we want to prove that the triangle $p_1p_2p_3$ contains an integer point on the line $x = 1$. The lower and upper points of the triangle in the line $x = 1$ are the two intersection points of $x = 1$ and each of the two segments p_1p_3 and p_2p_3. Their coordinates are respectively $(1, \frac{w}{v})$ and $(1, u+\frac{w-u}{v})$. Then the intersection of the line $x = 1$ and the triangle $p_1p_2p_3$ contains an integer point if and only if the interval $[\frac{w}{v}, \frac{uv+w-u}{v}]$ contains an integer namely if the interval $[w, w + u(v - 1)]$ contains a multiple of v, which is trivially true since there is necessarily a multiple of v in any interval $[w, w + v[$ and then in $[w, w + v - 1] \subset [w, w + u(v - 1)]$ as $u \geq 1$, and thus $u(v - 1) \geq v - 1$. □

The area of the jewel hull of S is finite unless all the points of S are colinear. However, in this case there exists a triangle with vertices in $\mathbb{R}^2$ that separates S from $\mathbb{Z}^2 \setminus S$.

The jewel hull consists of the intersection of a set of h halfplanes. Computing the vertices of the intersection of halfplanes is the dual [8, Chapter 8] of the computation of the convex hull of a given points set. In the general case, computing the intersection of h halfplanes takes $O(h \log h)$ time [8, Chapter 4]. However, since we already have the h halfplanes sorted by slope, we can use Graham scan [8, Chapter 1] to compute the jewel hull in $O(h)$ time. Notice that not all h halfplanes appear on the boundary of the jewel hull, which is the dual of the fact that some points may be in the interior of the convex hull.

3.2.2 Jewel Separation

The jewel separation is the final step to solve the `Edge minimization` problem. The jewel hull J has been computed and the problem is the polygonal separation of $IN = S$ and the jewel set $OUT' = Jewel(S)$. The previous step does not provide the set of jewels but the ordered list of edges of the jewel hull J as a sequence of linear equalities $\ell_i : a_i x + b_i y + c_i = 1$ with coprime integers a_i and b_i. An initial lattice point d_i of each given Diophantine straight line ℓ_i can be computed with the extended Euclid algorithm in $O(\log r)$ time. We can go from this first point to the other integer points of the line ℓ_i through translations of vectors $k\overrightarrow{(-b_i, a_i)}$ where $k \in \mathbb{Z}$. Nevertheless, J is a rational polytope. Its vertices are the intersection points of consecutive Diophantine lines ℓ_i but they are not necessarily integer points. It is even possible that some edges of the jewel hull do not contain any integer point. By computing the vertices of each edge e_i we can count all the jewels on ℓ_i and obtain a generating formula for them in $O(1)$ time and space for each edge. The jewels on ℓ_i are: $\bigcup_k d_i + k(-b_i, a_i)$. The computation of an integer point d_i per line ℓ_i for each one of our at most h Diophantine lines takes $O(h \log r)$ time. The computation of the vertices of J takes $O(h)$ time, and hence the computation of the formulas generating the jewels takes $O(h \log r)$ time and $O(h)$ space.

The jewels are determined in counterclockwise order according to their order of appearance in the jewel hull. Their cyclic index i goes from 0 to $|jewel(S)| - 1$. Furthermore, any pair of indices i, j with $i < j$ defines two intervals of indices, the interval $I_{i \to j}$ containing the indices of the successors of i until j and the interval $I_{j \to i}$ containing the indices of the successors of j

until i. We introduce now the precise meaning of *separation*. We say that a real line ℓ *separates* some jewels from S if S is located entirely on one side of ℓ while the jewels are located strictly on the other side. The fact that all jewels are located on the boundary of a convex polygon leads to the following simple lemma:

Lemma 11. *If ℓ is a line separating the jewels of indices i and j from S, then the line ℓ separates S from either the jewels with indices in $I_{i\to j}$ or the jewels with indices in $I_{j\to i}$.*

A naive approach to solve the polygonal separation problem of the sorted set of jewels from S is the following: Choose a starting jewel of index i_0. Search for the index j_0 such that the jewels with indices in the interval $I_{i_0\to j_0}$ can be separated from S and $|I_{i_0\to j_0}|$ is maximized. The method used to compute j_0 in constant time using our representation of the jewels will be detailed later. We then define i_1 as the successor of j_0 and repeat the process: search for j_1 such that $I_{i_1\to j_1}$ can be separated from S and the number of jewels in the interval is maximized. We repeat until we find an interval $I_{i_k\to j_k}$ which contains the predecessor of i_0. The number of lines of the solution is the number $k+1$ of intervals considered. This algorithm is illustrated Fig. 3.6. We call this greedy algorithm the `turn` routine since the strategy is to turn around the set S from a starting jewel p_{i_0}.

The difficulty of this approach is that different choices of the starting point p_{i_0} may lead to different numbers of separating lines (actually, they may differ by at most 1 line). The strategy to find the minimum number of separating lines is to test several starting jewels. Dynamic programming approaches might

Algorithm 1 turn(conv(S), Jewel(S),i_0)

Require: the convex hull conv(S), the ordered list of its jewels Jewel(S), and a starting jewel p of index i_0.

Ensure: A separating polygon with S inside and Jewel(S) outside.

1: Initialize i_0 as the index of the starting jewel, $k = 0$ and $I_{i_{-1}\to j_{-1}}$ as an empty interval

2: **while** $predecessor(i_0) \notin I_{i_{k-1}\to j_{k-1}}$ **do**

3: Compute j_k such that the jewels with indices in the interval $I_{i_k\to j_k}$ can be separated from S and $|I_{i_k\to j_k}|$ is maximized.

4: $i_{k+1} \leftarrow successor(j_k)$

5: $k = k + 1$

6: **return** The polygon obtained from the separating lines

be used to find an optimal solution as in [32], but in the framework of our `Edge minimization` problem in the lattice, we are able to obtain a major simplification.

The strategy to simplify the problem is the following. There are two families of jewels: the ones which chosen as starting jewel in the `turn` routine provide a minimal number of lines, their indices are denoted I_{OPT}, and the ones that provide a non optimal number of lines. Notice that if the index i_0 is in I_{OPT}, then all the indices i_k computed during the `turn` routine are also in I_{OPT} since it can be easily seen that they provide also optimal solutions. In the general case of polygonal separability, a large set of starting points has to be investigated until finding one leading to an optimal solution but in the framework of the separation of $IN = S$ and $OUT' =$ Jewel(S), we can provide

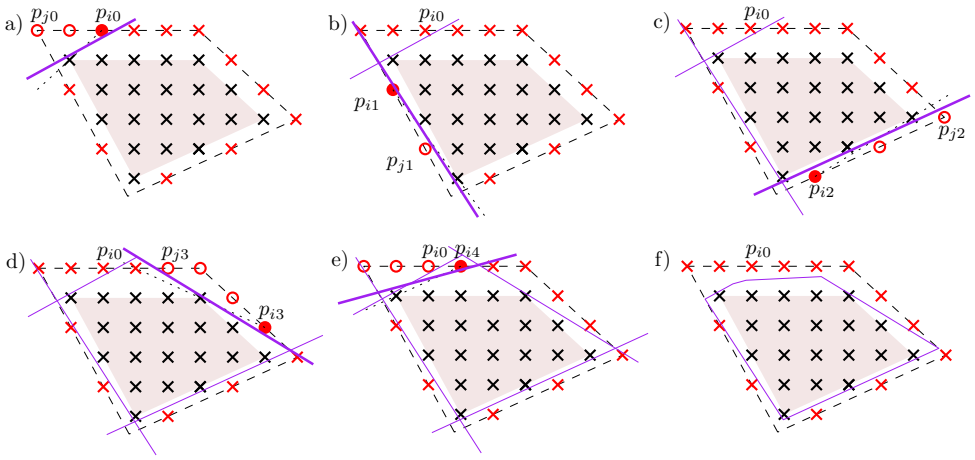

Figure 3.6 Turn algorithm. We start from a chosen starting jewel p_{i_0} and search for its last successor p_{j_0} that can be separated from S simultaneously with p_{i_0} by a single line. We then take the successor of p_{j_0} as new starting jewel p_{i_1} and search for the last successor p_{j_1} of p_{i_1} that can be separated with p_{i_1}... We repeat the process until reaching the predecessor of p_{i_0}.

a subset of at most 4 jewels containing at least one in I_{OPT}. It means that testing these four jewels as starting points of the `turn` routine is enough to find the optimal solution. The properties of the set I_{OPT} are presented in the next two lemmas.

The first lemma states that there is no line that simultaneously separates two jewels of a line ℓ_i and two jewels of ℓ_{i+1}.

Lemma 12. *Let ℓ_1 and ℓ_2 be two jewel lines. (i) If $\ell_1 \cap \ell_2 \notin \mathbb{Z}^2$ then there is no line that separates two jewels of ℓ_1 and two jewels of ℓ_2. (ii) If $\ell_1 \cap \ell_2 \in \mathbb{Z}^2$ then there is no line that separates three jewels of ℓ_1 and three jewels of ℓ_2.*

Proof. (i) Let order the jewels on ℓ_1: $J_1 = \{p_{1\_1}, p_{1\_2}, ...\}$ according to their distance to ℓ_2, and order the jewels on ℓ_2: $J_2 = \{p_{2\_1}, p_{2\_2}, ...\}$ according

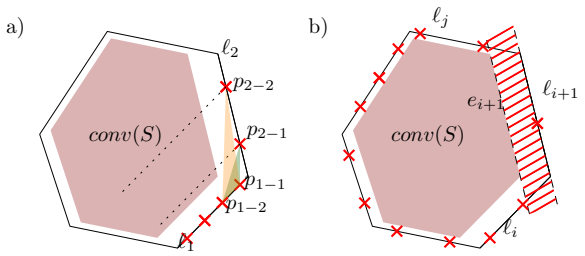

Figure 3.7 Jewel separation. a) If a single line separates both p_{1_2} and p_{2_2}, then the triangle $\triangle p_{1\_1}p_{1\_2}p_{2\_2}$ is larger than $\triangle p_{1\_1}p_{1\_2}p_{2\_1}$ and hence must contain a fourth lattice point, which is impossible. b) No jewels are located between e_{i+1} and ℓ_{i+1} hence it is impossible to separate simultaneously jewels from ℓ_i and jewels from ℓ_j.

to their distance to ℓ_1. Assume that there is a line l such that l separates two jewels of ℓ_1 and two jewels of ℓ_2 from conv(S). Then l separates $p_{1\_1}, p_{1\_2}, p_{2\_1}$ and p_{2_2} from conv(S). Hence the triangle $\triangle p_{1\_1}p_{1\_2}p_{2\_2}$ is located inside the jewel hull and outside of conv(S) (Fig. 3.7.a). As the triangle $\triangle p_{1\_1}p_{1\_2}p_{2\_1}$ is not degenerated we have $Area(\triangle p_{1\_1}p_{1\_2}p_{2\_1}) \geq \frac{1}{2}$. Hence the inequality $Area(\triangle p_{1\_1}p_{1\_2}p_{2\_2}) > Area(\triangle p_{1\_1}p_{1\_2}p_{2\_1})$ leads to $Area(\triangle p_{1\_1}p_{1\_2}p_{2\_2}) > \frac{1}{2}$. Using Pick's theorem we can conclude that $\triangle p_{1\_1}p_{1\_2}p_{2\_2}$ contains at least four lattice points. However, since $p_{1\_1}p_{1\_2}$ are two consecutive lattice points of ℓ_1, this means that there is a lattice point strictly inside the jewel hull and outside conv(S), which is impossible. Hence l does not exist. The proof of (ii) is the same, we just have to consider $p_{1\_0} = p_{2\_0} = \ell_1 \cap \ell_2$. □

We complete Lemma 12 with a lemma about the separation of jewels which are not in consecutive lines ℓ_i and ℓ_{i+1}.

Lemma 13. *If ℓ_i and ℓ_j are two non consecutive jewels lines: $j \geq i+2$, then there is no line that separates any jewel that belongs only to ℓ_i and any jewel that belongs only to ℓ_j.*

Proof. Consider ℓ_{i+1} and its associated edge on conv(S): e_{i+1}. By construction, there is no lattice point between ℓ_{i+1} and e_{i+1} (Fig. 3.7.b). Assume that there is a line l that separates jewels of both ℓ_i and ℓ_j As all the jewels belonging only to ℓ_i and all the jewels belonging only to ℓ_j are located on the same side s_j of e_{i+1} as S, l has to be in s_j to separate jewels of ℓ_i, then has to leave s_j in order to not intersect conv(S), and finally has to go back in s_j to separate jewels of ℓ_{i+1}. Hence l intersects e_{i+1} twice which is impossible. □

We now explain how to use Lemmas 12 and 13 to determine at most four jewels such that at least one of them leads to an optimal solution with the `turn` routine. In other words, we provide four indices with the guarantee that at least one of them is in I_{OPT}. For convenience, the successor of the index s is now simply denoted $s+1$ and so on with the successor of the successor denoted $s+2$. In the same manner, we also use $s-1, s-2, ...$ to denote the predecessors of s. When looking for a jewel in I_{OPT}, several cases might occur:

1. The jewel hull J has an edge e_i which does not contain any integer point. If we denote s the index of the first jewel after this edge, then I_{OPT} contains s. It is a corollary of Lemma 13. Considering an optimal solution, the vertex of index s cannot be included in the interval $I_{i_r \to j_r}$ containing $s-1$ because the interval would contain jewels of the lines ℓ_{i-1} and ℓ_{i+1} which is excluded by Lemma 13. Hence the index s is a

starting index namely an index of the form i_r of the considered optimal solution. As the indices i_r of the intervals $I_{i_r \to j_r}$ computed from an optimal starting index i_0 are also optimal, s is included in I_{OPT}.

2. The jewel hull J has an edge e_i with only one jewel s, hence I_{OPT} contains either s or $s+1$. Considering an optimal solution, it follows from Lemma 13 that $s-1$ and $s+1$ cannot be in an interval of the form $I_{i_r \to j_r}$ since they are on distant lines ℓ_{i-1} and ℓ_{i+1}. Hence, there exist either an index i_r equal to s or to $s+1$. It proves that one of these two indices s or $s+1$ is in I_{OPT}.

3. The jewel hull has an edge with only two jewels. Their indices are s and $s+1$. Considering an optimal solution, according to Lemma 13 the indices $s-1$ and $s+2$ cannot be in the same interval $I_{i_r \to j_r}$ because they belong to the distant lines ℓ_{i-1} and ℓ_{i+1}. Hence, there is at least a beginning of interval in s, $s+1$ or $s+2$. One of these three indices s, $s+1$, $s+2$ is in I_{OPT}

4. The edges of the jewel hull all contain at least three jewels. We choose any edge e_i and denote s, $s+1$, $s+2$ the indices of its three firsts jewels. According to Lemma 12 the indices $s+2$ and $s-2$ cannot be in the same interval $I_{i_r \to j_r}$. Hence, there is at least a beginning of interval in $s-1$, s, $s+1$ or $s+2$. One of these four indices $s-1$, s, $s+1$, $s+2$ is in I_{OPT}.

In any case, we can determine a set of at most four starting jewels with the guarantee that the `turn` algorithm provides an optimal solution for at least

one of them. We now explain how, in the turn algorithm 1, for a given jewel p_i we compute its last successor p_j that can be separated alongside him with a single line. Let p_i be on the jewel line ℓ_i, and let v_i be the end vertex of the edge of the convex hull parallel to ℓ_i. Consider the line $p_i v_i$. S is located on one side of $p_i v_i$, all the jewels that are located strictly on the other side can be separated alongside p_i (Fig. 3.6). It is clear that all jewels located on ℓ_i can be separated with p_i, and using Lemma 13 we know that the jewels located on ℓ_{i+2} cannot. Hence, all we have to do is determine the last jewel of ℓ_{i+1} that is located on the correct side of $p_i v_i$. This is easily done by computing the intersection point q of $p_i v_i$ and ℓ_{i+1} and expressing q as $d_{i+1} + \lambda(-b_{i+1}, a_{i+1})$ (We remind that the jewels on ℓ_{i+1} are expressed as: $\bigcup_k d_i + k(-b_i, a_i)$). From there a separating line can be computed by rotating slightly $p_i v_i$ around any points in between p_i and v_i.

The time complexity of the `turn` algorithm 1 is hence $O(h) = O(n^{1/3})$. This follows from the fact that h is an upper bound to the number of edges of the solution of the `Edge minimization` problem and $h = O(n^{1/3})$. Starting from any jewel, the algorithm computes a polygon that has at most one edge more than the optimal solution and each edge is computed in $O(1)$ time.

As the jewel hull is computed in $O(h \log r)$ time, the set of $O(1)$ starting jewels can be computed in constant time, and the `turn` algorithm 1 runs in $O(h)$ time. Hence the edge minimization algorithm, once provided with the convex hull of S, runs in $O(h \log r)$ time.

Algorithm 2 edge minimization(S)

Require: S a set of points.

Ensure: A minimal separating polygon if S is digital convex.

1: Test the digital convexity of S and compute conv(S) using quickhull

2: Compute the jewel hull of S using Graham scan

3: Compute at most four starting jewels

4: **for all** starting jewels **do**

5: Compute the minimal separating polygon using the given starting jewel using algorithm 1

6: **return** The minimal separating polygon

Chapter Four

Digital Convex Subsets

In this chapter, we develop algorithms to compute digital convex subsets. More precisely, we investigate a digital version of the potato peeling problem [15] that we call *digital potato peeling* and in which the goal is, given a set S, to find the largest digital convex subset of S. This problem has been stated in 2004, and a first heuristic given in 2005 [17].

In Section 4.1, we propose an embedding of the digital potato peeling problem in a directed acyclic graph (DAG) that leads to an $O(n^4 + n^2 \log r)$ algorithm to solve the digital potato peeling problem. Then, using the same strategy, we present an optimization of the algorithm that leads to a dynamic programming algorithm that runs in $O(n^3 + n^2 \log r)$ time.

In Section 4.2, we consider the *k-digital potato peeling* problem in which we look for the largest union of k subsets of S. Once again, we propose an embedding of the problem in a DAG, leading to an $O(n^{4k+1} + n^2 \log r)$ algorithm. We then show how a similar optimization as in Section 4.1 leads to a dynamic programming algorithm that runs in $O(n^{4k} + n^2 \log r)$ time.

In this chapter, we consider the largest digital convex subsets to be the

ones with the maximum number of points. However the algorithms described can easily be adapted to instead maximize the area, or even the convex hull's perimeter.

4.1 Digital Potato Peeling

In this section, we present an algorithm to solve the digital potato-peeling problem in $O(n^3 + n^2 \log r)$ time, where n is the number of input points and r is the diameter of the point set. We define the digital potato-peeling problem as follows:

Problem 3. *Digital potato-peeling*

Input: *A set $S \subset \mathbb{Z}^2$ of n lattice points given by their coordinates.*

Output: *The largest set $K \subseteq S$ that is digital convex (i.e., $\text{conv}(K) \cap \mathbb{Z}^2 = K$), where largest refers to $|K|$.*

We note that a digital convex set K can be described by its convex hull $\text{conv}(K)$, and that the vertices of $\text{conv}(K)$ are lattice points. In order to solve the digital potato-peeling problem, instead of explicitly building K, our algorithm constructs $\text{conv}(K)$. We also note that any convex polygon P with k vertices can be triangulated using $k-2$ triangles that share a common vertex, the bottom-most vertex ρ of P for instance. We name such a triangulation a *fan triangulation* (Fig. 4.1). In order to solve problem 3 we first consider the following rooted variation of the digital potato-peeling problem, where the point ρ has been given as part of the input.

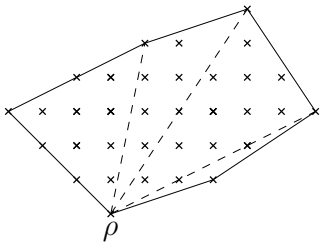

Figure 4.1 Fan triangulation. A digital convex set, its convex hull, and a fan triangulation from its bottom-most point ρ

Problem 4. *Problem* `Rooted digital potato-peeling`

Input: *A set $S \subset \mathbb{Z}^2$ of n lattice points given by their coordinates. We define ρ as the bottom-most point in S.*

Output: *The largest set $K \subseteq S$ that is digital convex and has ρ as the right-most point at the bottom-most row of K.*

The difference between the digital potato-peeling problem and its rooted variation is that in the digital potato peeling the bottom-most point is not necessarily in the solution, whereas the bottom-most point is forced to be in the solution in the rooted version of the problem. The idea we develop in this section consists in solving the digital potato peeling problem by testing all possible roots for the rooted version of the problem.

In the rooted version of the problem, we only have to consider points in S that are located either above ρ or on the same row as ρ to its left. We refer to this subset of S as S_ρ, and we refer of ρ as the *root*. Let $p_1, \ldots, p_n$ denote the points of S_ρ sorted clockwise around ρ, starting from the left.

Let $\triangle_{i,j}$ denote the (closed) triangle whose vertices are ρ, p_i, p_j with $i < j$. We say that a triangle $\triangle_{i,j}$ is *valid* if $\triangle_{i,j} \cap \mathbb{Z}^2 = \triangle_{i,j} \cap S$, that is if all the lattice points inside $\triangle_{i,j}$ are in S. To algorithmically verify that $\triangle_{i,j}$ is valid,

we compare $|\triangle_{i,j} \cap S|$ and $|\triangle_{i,j} \cap \mathbb{Z}^2|$. The value of $|\triangle_{i,j} \cap \mathbb{Z}^2|$ is determined as follows. Pick's theorem states that the area of a triangle with lattice vertices is equal to $n_b/2+n_i-1$, where n_b is the number of boundary lattice points and n_i is the number of interior lattice points. The value of n_b can be computed using a GCD computation for each edge in $O(\log r)$ time, where r is the diameter of the triangle. Plugging in the area of the triangle, we obtain the number of lattice points $|\triangle_{i,j} \cap \mathbb{Z}^2| = n_b + n_i$. To compute $|\triangle_{i,j} \cap S|$ we use a triangle range counting query described in Section 2.3.1. We remind that those queries can be answered in $O(1)$ time after preprocessing S in $O(n^2)$ time [30]. Hence, the total time to test the validity of a triangle (after preprocessing) is $O(\log r)$. However, on the account that there is $O(n^3)$ triangles and only $O(n^2)$ edges, we preprocess the number of lattice points on each edge in $O(n^2 \log r)$ time and are able to test the validity of triangle in $O(1)$ time after a preprocessing of $O(n^2 \log r)$ time.

We now consider K_i, a digital convex subset of S_ρ whose root is ρ. We can build $\text{conv}(K_i)$ by appending the triangles of its fan triangulation clockwise. All the triangles used have lattice vertices, are valid, and their bottom-most vertex is ρ. Now; in order to solve the rooted digital potato-peeling problem, we want to find the appending that results in the largest K_i possible.

4.1.1 Directed Acyclic Graph embedding

We start by showing how to find the largest digital convex subset K_i by building a DAG $\mathcal{G}$ that represents all the possible ways to append rooted triangles in a convex manner. Then, in a second time, we will present a faster dynamic

programming algorithm which is based on the same construction. In order to avoid confusion with the geometric terms, we will refer to the graph vertices as *nodes*, and the graph edges as *arcs* (Fig. 4.2).

$\mathcal{G}$ is built in the following way:

- Each node $\mathcal{N}_i$ of $\mathcal{G}$ represents a valid rooted triangle. Analogously, all valid rooted triangles are represented by a node in $\mathcal{G}$

- There is an arc $\mathcal{A}_{ij}$ from the node $\mathcal{N}_i$ towards the node $\mathcal{N}_j$ if and only if:

  (i) The union of the triangle $\triangle_i$ associated to $\mathcal{N}_i$ and the the triangle $\triangle_j$ associated to $\mathcal{N}_j$ is a convex quadrilateral q_{ij}, and

  (ii) $\triangle_i$ is the first triangle in the clockwise rooted fan triangulation of q_{ij}.

- The weight of an arc $\mathcal{A}_{ij}$ is equal to the number of lattice points in $\triangle_j$ not in $\triangle_i$, that is the number of lattice points in $\triangle_j$ minus the number of lattice points on the edge shared by $\triangle_i$ and $\triangle_j$.

- A starting node $\mathcal{A}_\emptyset$ is added to $\mathcal{G}$

- Initialization arcs $\mathcal{A}_{\emptyset,i}$ are added from the starting node towards each other node $\mathcal{N}_i$. Their weight is equal to the number of lattice points in $\triangle_i$

Solving the rooted digital potato-peeling problem is equivalent to finding the longest path in $\mathcal{G}$. This property is a corollary of the following lemma.

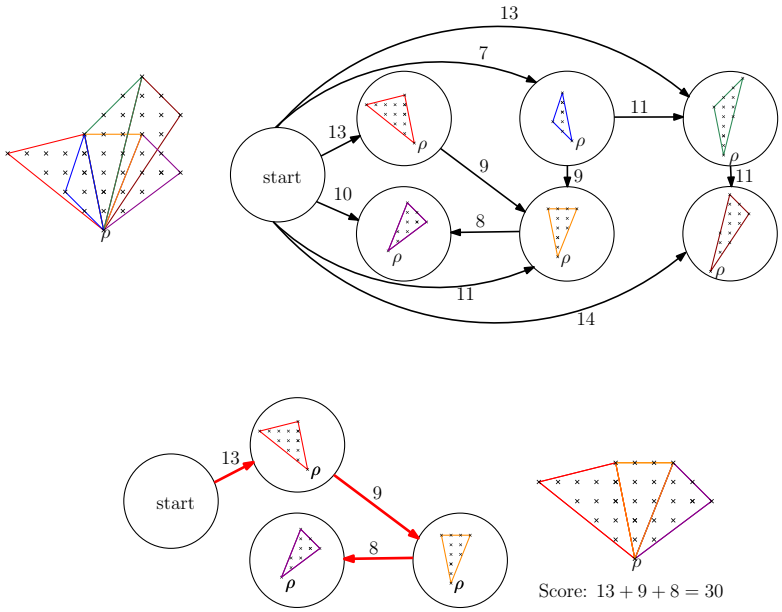

Figure 4.2 Rooted peeling graph. Example of a DAG built to solve the rooted peeling problem for 6 valid triangles. For two triangles to be connected by an arc in the graph, they need to have a common edge that starts from ρ, and their union must be a convex polygon. The weight of each arc is equal to the number of lattice points inside the right-most triangle that are not located on the common edge between the two triangles.

Lemma 14. *(i) Every path in $\mathcal{G}$ represents a digital convex set. The length of the path is equal to the number of lattice points in the digital convex set.*

(ii) Each digital convex subset of S is represented by a path in $\mathcal{G}$

Proof. Let $\triangle_i$ be the triangle represented by the node $\mathcal{N}_i$ in $\mathcal{G}$. Note that ρ is a vertex of $\triangle_i$. Let p_1 and p_2 be the two other vertices of $\triangle_i$ such that the points $\rho p_1 p_2$ are oriented clockwise. We call ρp_1 the *left edge* of $\triangle_i$, and ρp_2 the *right edge*. By construction, all arcs starting from $\mathcal{N}_i$ end into a node representing a triangle whose left edge is ρp_2. Furthermore, if we denote p_3

the third vertex of this triangle, we have the point $p_1p_2p_3$ oriented clockwise. Hence, every path $(\mathcal{N}_\emptyset, \mathcal{N}_{i_1}, \mathcal{N}_{i_2}, \mathcal{N}_{i_{k-1}})$ in $\mathcal{G}$ represents the fan triangulation of a polygon $P = (p_1, p_2, ..., p_k)$ such that the points $\rho, p_1, p_2, ..., p_k$ are all in convex positions. As only the valid triangles are represented in $\mathcal{G}$, all lattice points inside P are in S, and hence $P \cap S$ is digital convex. The length of the path is equal to the sum of the number of lattice points inside each triangle, minus the lattice points on the edges: $(\rho p2, \rho p3, ..., \rho p_{k-1})$, which is equal to the number of lattices inside P (Fig. 4.2). This proves (i).

Let P be the convex hull of a digital convex subset of S. Let $\triangle_1\triangle_2...\triangle_k$ be the rooted fan triangulation of P. By construction, for each i from 1 to k we know that $\triangle_i$ is represented by $\mathcal{N}_i$ in $\mathcal{G}$, and for each i from 1 to $k-1$, we know that there is an arc from $\mathcal{N}_i$ to $\mathcal{N}_{i+1}$. Hence, there is a path in $\mathcal{G}$ representing P. This proves (ii).

□

Computing the longest path in the DAG $\mathcal{G}$ takes linear time in the number of arcs in $\mathcal{G}$. The number of nodes in $\mathcal{G}$ is at most $O(n^2)$, and each of these nodes has at most n incoming arcs. Hence the number of arcs in $\mathcal{G}$ is $O(n^3)$ and the rooted digital potato peeling problem can be solved in $O(n^3)$ time after $O(n^2 \log r)$ time preprocessing.

Now, in order to solve the digital potato-peeling problem, we solve the rooted digital potato-peeling for each of the n possible roots. As the preprocessing is common to each instance of rooted potato-peeling problem, we only have to preprocess once. Which leads to an $O(n^4 + n^2 \log r)$ algorithm to solve the digital potato-peeling problem.

4.1.2 Dynamic programming

We now show that the same idea of fan triangulation and clockwise triangles appending can lead to a dynamic programming algorithm that runs in $O(n^3 + n^2 \log r)$ time. This algorithm makes use of the same precomputation as the DAG one we just presented.

Once again, we present an algorithm that solves the rooted potato-peeling problem, and we then use this algorithm to solve the potato-peeling problem by trying every root possible.

The algorithm incrementally builds the fan triangulation of $\text{conv}(K)$ by appending valid triangles in clockwise order using dynamic programming. At each step, we ensure the digital convexity through the following property. Let $\text{conv}(K')$ be the convex hull of a digital convex set K' rooted at ρ with $\triangle_{h,i}$ as the right-most triangle. If $\triangle_{i,j}$ is valid and $\triangle_{h,i} \cup \triangle_{i,j}$ is convex, then $K' \cup (\triangle_{i,j} \cap \mathbb{Z}^2)$ is digital convex.

First, we sort all the points around ρ clockwise. The idea of the algorithm is the following: for all pairs of points $p_i, p_j \in S_\rho$ with $i < j$ such that $\triangle_{i,j}$ is valid, we want to compute the largest convex polygon amongst that have $\triangle_{i,j}$ as their last triangle in the clockwise rooted fan triangulation. We refer to this largest convex polygon as $C_{i,j}$. The key property to efficiently compute $C_{i,j}$ is

$$C_{i,j} = \triangle_{i,j} \cup \max_h C_{h,i}, \text{ where } h < i \text{ is such that } \triangle_{i,j} \cup \triangle_{h,i} \text{ is convex.}$$

In order to compute $C_{i,j}$, we do the following. For each value of i, we sort all valid $\triangle_{i,j}$ for $j > i$ in counter-clockwise order of p_j around p_i into a list of triangles $\mathcal{T}_i$, obtaining n lists of $O(n)$ triangles each. Sorting all

angular sequences can be done in $O(n^2)$ time using the dual[8, Chapter 11]. Moreover, testing the validity of each triangle takes $O(1)$ time, which gives a total running time of $O(n^2)$. Next, we explain the dynamic programming part of the algorithm.

For each i from 1 to n, we do the following. First, we sort all values of $C_{h,i}$ for $h < i$ by decreasing value into a list $\mathcal{C}_i$. Then, we consider every $\triangle_{i,j}$ in the order given by $\mathcal{T}_i$, testing each $C_{h,i}$ from the largest area to the smallest. If $C_{h,i} \cup \triangle_{i,j}$ is convex, then we set $C_{i,j} = C_{h,i} \cup \triangle_{i,j}$. Otherwise, we permanently remove $C_{h,i}$ from the list $\mathcal{C}_i$ (Fig. 4.3). Next, we justify the correctness of this procedure, especially the fact that we are allowed to remove the aforementioned values of $C_{h,i}$ from $\mathcal{C}_i$.

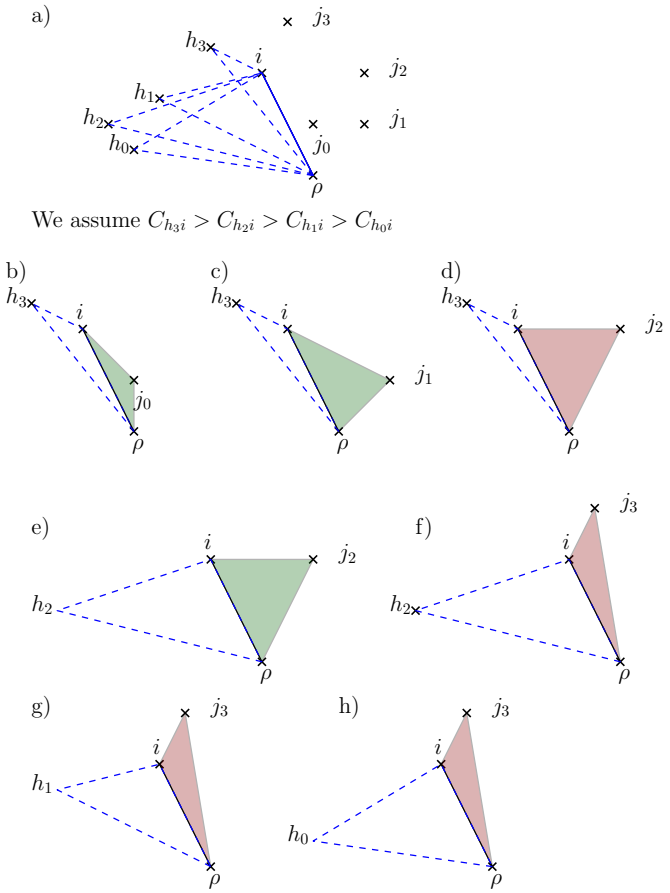

Figure 4.3 Dynamic algorithm for the rooted peeling. a) At this step, we consider the point i .The four largest convex polygons ending at the point i: $C_{h_0 i}$, $C_{h_1 i}$, $C_{h_2 i}$, and $C_{h_3 i}$ were previously computed by the algorithm. They are sorted according to their respective score. We will consider them in order from largest to smallest. In this step, we are computing C_{ij_0}, C_{ij_1}, C_{ij_2}, and C_{ij_3}. The points j_0, j_1, j_2, and j_2 are sorted around i. We will treat them in their sorted order. b) We test $\rho i j_0$ with the largest polygon $C_{h_3 i}$. Their union is a convex polygon therefore $C_{ij_0} = C_{h_3 i} \cup \rho i j_0$. c) We test $\rho i j_1$ with $C_{h_3 i}$. Their union is a convex polygon therefore $C_{ij_1} = C_{h_3 i} \cup \rho i j_1$. d) We test $\rho i j_2$ with $C_{h_3 i}$. Their union is not a convex polygon. We forget about $C_{h_3 i}$ for the remainder of the step, and we move on to the next largest convex polygon. e) We test $\rho i j_2$ with $C_{h_2 i}$. Their union is a convex polygon therefore $C_{ij_2} = C_{h_2 i} \cup \rho i j_2$. f) We test $\rho i j_3$ with $C_{h_2 i}$. Their union is not a convex polygon. g) We test $\rho i j_3$ with $C_{h_1 i}$. Their union is not a convex polygon. h) We test $\rho i j_3$ with $C_{h_0 i}$. Their union is not a convex polygon. As there is no convex polygons left we can conclude that $C_{ij_3} = \rho i j_3$.

Since we sorted the list $\mathcal{T}_i$ of $\triangle_{i,j}$ counter-clockwise by p_j around p_i, we have the following key property. For each $\triangle_{i,j}$ preceding $\triangle_{i,k}$ in the ordering $\mathcal{T}_i$, we have that for all $h < i$ if $C_{h,i} \cup \triangle_{i,j}$ is not convex, then $C_{h,i} \cup \triangle_{i,k}$ is not convex either. Hence, the values of $C_{h,i}$ removed from $\mathcal{C}_i$ cannot form a convex polygon with the triangles $\triangle_{i,j}$ that appear later in the list $\mathcal{T}_i$.

The running time of the dynamic programming part for each value of i is the following. First, we retrieve the angular sorted list of valid triangles $\mathcal{T}_i$. This step only takes $O(n)$ time as we preprocess the angular sorting of all points around i. The remaining part also takes $O(n)$ time since at each step, we either remove a convex polygon $C_{h,i}$ from $\mathcal{C}_i$ or we advance through the list of triangles $\mathcal{T}_i$. Considering all n values of i and the initial sorting, the total time to solve Problem 4 is $O(n^2)$. In order to solve Problem 3, we test all n possible values of $\rho \in S$, proving the following theorem.

Theorem 15. *There exists an algorithm to solve Problem 3 (digital potato peeling) in $O(n^3 + n^2 \log r)$ time, where n is the number of input points and r is the diameter of the input.*

4.2 k-Digital Potato Peeling

In this section we give interest and present an algorithm to solve the k-digital potato-peeling problem that we define as follows.

Problem 5. *k-digital potato-peeling*

Input: *A set $S \subset \mathbb{Z}^2$ of n lattice points given by their coordinates, and a value $k \in \mathbb{N}$.*

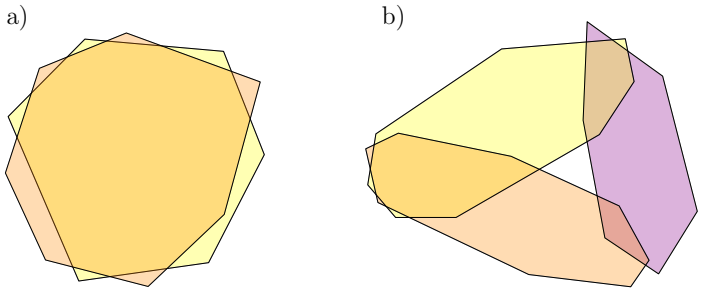

Figure 4.4 k-peeling solutions. The convex hulls can intersect any number of times as shown in a). Furthermore, for 3 or more polygons, holes can appear as shown in b).

Output: *k subsets $K_i \in S$ that are digital convex (i.e., $\mathrm{conv}(K) \cap \mathbb{Z}^2 = K$), such that $K = \bigcup_i K_i$ is the largest, where largest refers to $|K|$.*

Note that the 1-digital potato peeling problem is the potato peeling problem that we solved in $O(n^3+n^2 \log r)$ time in Section 4.1. In 2018, we published a specific algorithm for the 2-digital potato peeling [24], its running time is roughly $O(n^9)$. Here we present the first algorithm that solves the more general k-digital potato peeling. Its running time is $O(n^{4k} + n^2 \log r)$, which improves our previous best known algorithms for the 2-potato peeling problem [24]. The k-potato peeling introduces some new complications compared to the simpler digital potato peeling. Some of those complications come from the fact that the convex hull of two sets in a solution can intersect an arbitrarily large number of times, or that holes can be present in the middle of the union of several convex hulls (Fig. 4.4).

Once again, as in Section 4.1, we will characterize a digital convex subset by its convex hull. We note that any convex hull can be separated into two x-monotone polylines that we respectively name the *upper hull*, which is the

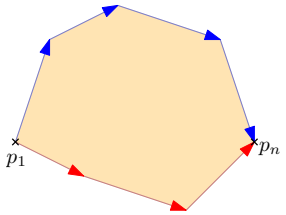

Figure 4.5 Upper and lower hull. The upper hull (in blue) and the lower hull (in red) that both go from the leftmost point(p_1) to the rightmost point (p_n).

part of the convex hull going from the leftmost point to rightmost point in clockwise order, and its counter clockwise counterpart that we name the *lower hull* (Fig. 4.5).

For two x-monotone polylines PL_1, PL_2 both going from p_1 to p_n to represent a convex hull, it is necessary and sufficient that one polyline always turns right and that the other polyline always turns left. Using this representation, we can incrementally build any convex polygon. While incrementally building the upper and lower hulls, we can ensure the convexity by simply testing the orientation of the angle when appending edges to the upper or lower hull. This means that during the construction of the hull, and at any time only the last appended edge needs to be known for each partial half hull. The construction is finished once both half hulls ends at the same point.

4.2.1 Computing the number of points inside the hull

The previous method, allows to build a set of edges that represent a convex polygon P while only maintaining the knowledge of 2 edges at any time. In this section, we show how we can use this method to compute the surface inside P while only maintaining the knowledge of 2 edges at any time.

Unlike the appending done in Section 4.1 in order to solve the digital potato peeling problem, the method we just described does not directly append surfaces together, but edges. Hence, using the method as is, the number of points inside P cannot be computed. In order to do so, we add the following ordering rule during the construction.

When choosing whether we append an edge to the top hull or to the bottom hull, we chose to only append an edge to the *least advanced* of the two currently known edges. We define the *least advanced* edge as follows:

Given two edges e_1 and e_2 respectively delimited by the points $(p_1(x_1, y_1)$, $p_2(x_2, y_2))$, and the points $(p_3(x_3, y_3)$, $p_4(x_4, y_4))$, such that $xmax_{e1} = Max(x_1, x_2)$ and $xmax_{e2} = Max(x_3, x_4)$. e_1 (resp. e_2) is the least advanced edge if and only if $xmax_{e1} \leq xmax_{e2}$ (resp. $xmax_{e2} \leq xmax_{e1}$). In other words, the least advanced edge is the one with the smallest maximal x-coordinate. This definition can be extended to more than two edges.

We now explain how this least advanced edge rule allows us to effectively compute the surface of the convex polygon P we are building, while only maintaining the knowledge of at most two edges at a time. We call *horizontal span* of any geometrical object O the smallest interval I such that for each point $p(x, y) \in O$, $x \in I$.

At each step of the computation only two edges are known: one for the upper hull and one for the lower hull. We define I_u (resp. I_l) to be the horizontal span of the currently known upper edge (resp. lower edge), and $I = I_u \cap I_l$. Let H_u (resp. H_l) be the half plane that is located below the upper edge (resp. above the lower edge), and let V be the vertical strip of

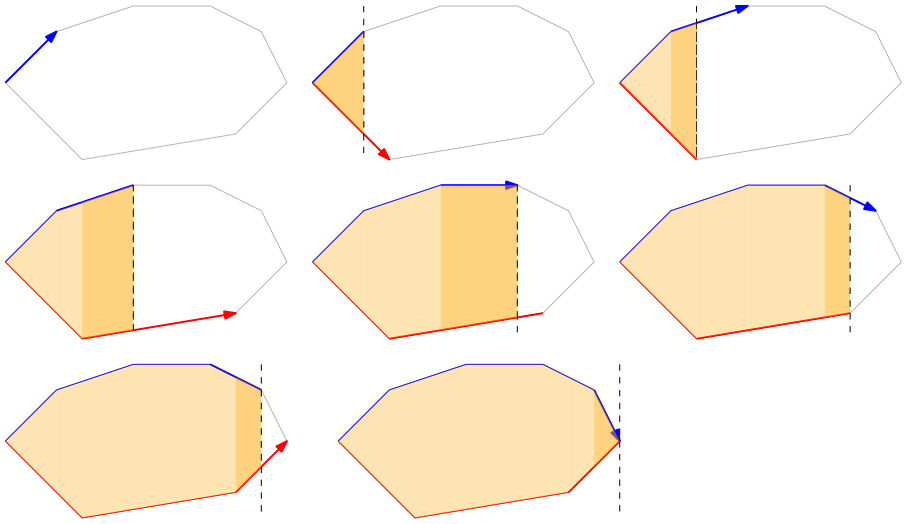

Figure 4.6 Edge advancing. Of the two edges, only the least advanced one moves forward. Then the surface of the trapezoid (shown in dark orange) located between the upper edge and the lower edge is computed.

∞ height whose horizontal span is I. In order to compute the surface inside P, what we do is that when appending an edge, we compute the trapezoid defined as: $H_u \cap H_l \cap V$ (Fig. 4.6).

P is equal to the union of all the trapezoid, in deed: Consider any edge e_u of the upper hull. By only appending an edge to the least advanced half hull we ensure that for every edges of the lower hull e_l such that e_u and e_l have an intersecting horizontal span, both e_u and e_l, at some point during the construction, will be known at the same time. Hence the entirety of the surface inside the convex hull will be considered, without any overlap.

As P is equal to the union of all the trapezoid, and as the intersection on any two adjacent trapezoid is a segment that is known during the computation, we can compute the area, or the number of lattice point inside P.

We now generalize this method in order to compute the surface of the union of any fixed number k of convex polygons. This time the construction requires us to know $2k$ edges at any time, that is 2 edges for each polygon, one for each half hull. The idea stays exactly the same as for one convex polygon. At each step of the computation, only the least advanced of the $2k$ edges is allowed to advance, and we only compute the area in the vertical strip of common x-coordinates to all the $2k$ edges. However, a special attention must be paid to polygons that have not started yet, or have already been finished. Those two situations are detected and dealt with in the following manner: if the upper edge and the lower edge of a same convex polygon start (resp. end) at the same point, this means that those are the starting (resp. ending) edges of said polygon. In this case we consider that the abscissa of those edges spans starts from $-\infty$ (resp. goes until ∞)) when computing the common horizontal span. We know that all convex polygons have been computed when, for each convex polygon, the upper edge and the lower edge end at the same point (Fig. 4.7).

Now, we will show how using the aforementioned representation, we can build the succession of edges (that we call *convex path*) that describes and allows to compute the cardinality of any union of k digital convex polygons in S, while working with only $2k$ edges at a time. In order to solve the k-potato peeling problem, we want to find the convex path that leads to the largest union of k digital convex sets. We call *partial convex path* the beginning of any convex path. Unlike a convex path, the end upper edges and lower edges of a partial path does not necessarily meet.

An upper edge u_e and a lower edge l_e are said *compatible* if there is a convex

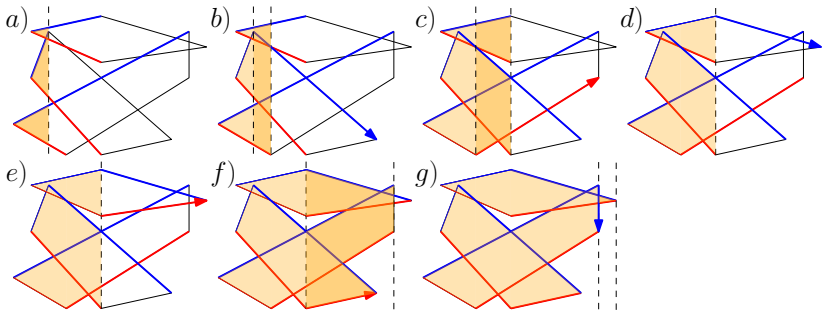

Figure 4.7 Visualisation of the k-peeling algorithm. Only the surface in the abscissa common to all edges is computed. When the two edges of a same convex polygon start from the same point (as seen in a)), we extend the abscissa of those edges from $-\infty$. Similarly when the two edges of a same convex polygon end at the same point (as seen in f) and g)), we extend the abscissa of those edges towards ∞.

polygon P such that in the construction we previously described both u_e and l_e can appear at the same time under the least advanced constraint(Fig. 4.8), that is if:

- the lower edge is located entirely within the half plane below the upper edge, and if
- the upper edge is located entirely within the half plane above the lower edge, and
- if the lower edge and the upper edge have at most one common point (namely the starting or ending point), and if
- the intersection of their horizontal span different to $\emptyset$, and if
- the intersection of the convex hull of the two edges with S is digital convex.

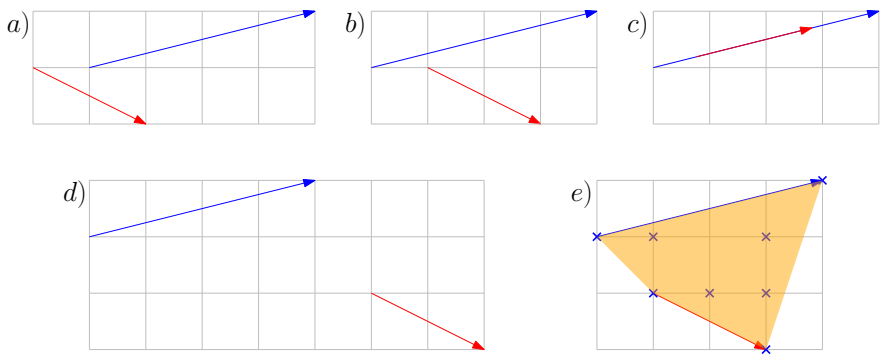

Figure 4.8 Compatible edges. In a) the lower edge is not located below the upper edge. In b) the upper edge is not located above the lower edge. In c), the upper and lower edge have more than one point in common. In d) the horizontal spans of the lower and upper edge are distinct. In e) The intersection of S (shown in blue crosses) with the trapezoid is different from the intersection of $\mathbb{Z}^2$ with the trapezoid. One lattice point is missing.

Furthermore, in the context of the k-digital potato peeling, in order to ensure that the result polygon P is not only convex, but also represents a digital convex subsets of S, we add the constraint that $\text{conv}(u_e, l_e) \cap \mathbb{Z}^2 \subset S$.

We now present two algorithms that solve the k-digital potato peeling problem using the aforementioned construction to describe a union of k convex polygons. The first algorithm embeds the problem in a *DAG* and runs in roughly $O(n^{4k+1})$ time. The second algorithm relies on dynamic programming for a total running time of roughly $O(n^{4k})$.

4.2.2 Using a DAG

The strategy is to encode the problem in a DAG $\mathcal{G}(V, E)$ whose longest path corresponds to the solution of the problem. Once again, to avoid confusion, we use the terms *node* and *arc* when referring to a DAG and keep the terms

vertex and *edge* when referring to polygons.

We build $\mathcal{G}$ in the following manner. Each node of $\mathcal{G}$ represents k ordered pairs of compatible edges. Within a node, each pair of compatible edge represents the upper edge and the lower edge of a convex polygon. As there are $O(n^4)$ pairs, we have at most $O(n^{4k})$ nodes in $\mathcal{G}$.

Let $N(C_1(u_1, l_1), C_2(u_2, l_2), ..., C_n(u_n, l_n))$ denote a node of $\mathcal{G}$, where $C_i(u_i, l_i)$ represent a pair of compatible edges, and where u_i (resp. l_i) denotes the upper edge (resp. lower edge) associated to the representation of the i^{th} convex polygon.

We put an arc from the node N_1 towards the node N_2 if $\exists j \in [1..n]$ such that $\forall_{i \in [1..n], i \neq j} : N_1(C_i) = N_2(C_i)$, and $N_1(C_j(u_j)) = N_2(C_j(u_j))$, and $N_1(C_j(l_j))$ ends at the same points that $N_2(C_j(l_j))$ starts, and such that $N_1(C_j(l_j))$ and $N_2(C_j(l_j))$ forms a convex angle. We also put an arc between from N_1 towards N_2 if $\exists j \in [1..n]$ such that $\forall_{i \in [1..n], i \neq j} : N_1(C_i) = N_2(C_i)$, and $N_1(C_j(l_j)) = N_2(C_j(l_j))$, and $N_1(C_j(u_j))$ ends at the same points that $N_2(C_j(u_j))$ starts, and such that $N_1(C_j(u_j))$ and $N_2(C_j(u_j))$ forms a convex angle. In more simple terms, we put an arc between two nodes if and only if only one edge changes between these two nodes, the changed edge has to be the least advanced one in order to respect the least advanced rule, and the new edge has to be compatible with both: the edge it replaces (preserving convexity), and its associated lower/upper edge.

As there are at most n arcs exiting a node, there are at most $O(n^{4k+1})$ total arcs in $\mathcal{G}$.

We now explain how we weight the nodes in $\mathcal{G}$. Let I be the interval

corresponding to the common horizontal span of all the edges of a given node N. Let S_i be the surface in between the upper edge $N(C_i(u_i))$ and the lower edge $N(C_i(l_i))$ of the i^{th} convex polygon within the horizontal span defined by I (Fig. 4.6). The weight of a given node N is equal to the number of lattice points in: $S_{union} = \bigcup_{i \in [1..k]} S_i$ minus the number of lattice points in S_{union} whose x-coordinate are equal to the minimal x-coordinate in S_i, unless the lattice point is the starting point of both the currently known edges of the same convex polygon (and hence is the starting point of both partial hull). This represents the lattice points inside the surface of the intersection of all the different convex polygons in the common horizontal span minus those that where already accounted for in the previous node.

We add to $\mathcal{G}$ a start node and an end node. There is an arc from the start node to N if $\forall i$ $N(C_i(u_i))$ and $N(C_i(l_i))$ start from the same point, that is if all convex polygons are still being represented by their first pair of edge. Similarly, there is an arc from N towards the end node if $\forall i$ $N(C_i(u_i))$ and $N(C_i(l_i))$ end at the same point, that is if all convex polygons are being represented by their last pair of edge.

We now explain, in Lemma 16 why each path in $\mathcal{G}$ represents the union of k digital convex sets and why the number of lattice points of the intersections of the convex hull of those sets is properly computed.

Lemma 16. *Every path in $\mathcal{G}$ going from the start node to the end node represents the union of k digital convex sets. And the length of the path is equal to to the number of lattice points inside the union of those k digital convex sets.*

Proof. We consider any path in $\mathcal{G}$ that goes from the start node to the end

node. The first node visited after the start node gives, two edges for each one of the k convex polygons we are building. For each polygon, these two edges are the start of the bottom hull and the start of the top hull. From now on, thanks to the way the arcs were added to the graph, each node visited, to the exception of the end node, will change exactly one edge to one polygon in the representation of the k convex polygons. This edge is added in such a way that ensures convexity with the one it replaces. As for each of the k polygons P_i, their top hull T_i and bottom hull B_i do start at the same point s_i and end at the same point e_i, we know that they do represent a convex polygon.

Now, the fact that the number of lattice points in the union of the k convex polygons is properly computed comes from the fact that:

- At each node, we only considered the surface in the common horizontal span (A special attention is given to the left limit of this horizontal span). This ensure that no overlap exists between two different nodes of the same path.

- Each arc only allows to change the least advance edge. This makes sure that, for every node, the common horizontal span is adjacent to the common horizontal path of its predecessor. This ensure that all the surface of the union of the convex polygons is considered (Fig. 4.7).

Finally, as only pairs of compatible edges are present in every node of $\mathcal{G}$, we know that every convex polygon represented in a path in the graph is a digital convex subset of S.

□

Furthermore, as all compatible edges are considered in the graph, naturally all unions of k convex digital subsets are indeed represented by a path in $\mathcal{G}$, and hence finding the longest path in $\mathcal{G}$ is equivalent to solving the k-digital potato peeling problem. Since there are $O(n^{4k+1})$ arcs in $\mathcal{G}$ this part of the algorithm takes $O(n^{4k+1})$ time. In addition, in order to compute $\mathcal{G}$, the algorithm requires to compute the compatible pair of edges. This can be done by iterating on all of the $O(n^4)$ pairs of edges, and testing whether their convex hull represents a digital convex subset of S or not. This test takes $O(1)$ time after a preprocessing time of $O(n^2 \log r)$ required in order to compute the number of lattice point on each edge, and an extra preprocessing time of $O(n^2)$ required for the triangular range counting algorithm presented in Section 2.3.1. Solving the k digital potato peeling using a DAG hence result in a time complexity of $O(n^{4k+1} + n^2 \log r)$ using $O(n^{4k+1})$ space.

4.2.3 Dynamic programming

We can use the same strategy that we used in Section 4.1 that allowed us to go from an $O(n^4)$ time algorithm using a DAG representation to roughly an $O(n^3)$ time algorithm using dynamic programming. This way, we can propose a roughly $O(n^{4k})$ time algorithm to solve the k-digital potato peeling problem.

To do so, we consider the following sub-problem:

Problem 6 (partial k-digital potato peeling)**.** *Given k pairs of compatible edges, what is the maximal partial convex path that has those edges as currently known edges.*

The answer to the k-digital potato peeling problem is the largest answer to the partial k-digital potato peeling problem amongst all the $O(n^{3k})$ pairs of edges such that for each pair of edges the upper edge and the lower edge meet at the same end point. The algorithm we propose dynamically computes the solution to the partial k-digital potato peeling problem for each of the $O(n^{4k})$ pairs of edges of S.

First we compute all the compatible pairs of edges. To do so, for each of the $O(n^4)$ pairs of edges within S we test each condition mentioned in the definition of compatible edges. In order to test the digital convexity of the convex hull of the two edges, we test the following equality: $|Q \cap S| = |Q \cap \mathbb{Z}^2|$. First using Pick's formula, we compute $|Q \cap \mathbb{Z}^2|$. This takes $O(1)$ time after $O(n^2 \log r)$ preprocessing time, where r is the diameter of S. The preprocessing is required in order to compute the number of lattice point located on each edge. Then, we compute $|Q \cap S|$ in $O(1)$ time after $O(n^2)$ preprocessing time using the method described in Section 2.3.1. Hence we can determine whether or not $Q \cap S = Q \cap \mathbb{Z}^2$ in $O(1)$ time after $O(n^2 \log r)$ preprocessing time.

Once all the compatible pair of edges are computed, we sort all the points of S, from left to right. Let p_i denote the i^{th} point of S in this sorted order. We then iterate on the p_i in their sorted order and will consider the partial path that ends at p_i. We hence consider two distinct possibility: p_i is in a upper edge and p_i is in a lower edge. As those two situations are treated with the exact same approach, we only describe the case where p_i in an upper edge. At this point, we fetch every point p_j, $j > i$ and sort them around p_i counter-clockwise, we also fetch all the points p_h, $h < i$ and sort them around

p_i clockwise. This order gives us the following property which will be useful later.

Lemma 17. *For any given pair of points p_h and p_j such that $h < i < j$: if $p_h p_i p_j$ turns right then:*

- *For each p'_h preceding p_h in the ordering $p'_h p_i p_j$ turns right*
- *For each p'_j preceding p_j in the ordering $p_h p_i p'_j$ turns right*

also if $p_h p_i p_j$ does not turn right then:

- *For each p'_h following p_h in the ordering $p'_h p_i p_j$ does not turn right*
- *For each p'_j following p_j in the ordering $p_h p_i p'_j$ does not turn right*

We then fetch the $O(kn^{4k-1})$ previously computed largest partial convex paths that have p_i as one of their upper hull end point of an upper edge, such that p_i is the left most of all the $2k$ end points of edges, and such that all the pair of edges are compatible. We split those $O(kn^{4k-1})$ partial convex paths into $O(n^{4k-2})$ lists, so that all the members of a same list have their $4k-2$ points that are not part of the upper edge that ends with p_i identical.

For a given partial convex path, let p_h be the point located on the same upper edge as p_i. The lists are sorted according to the position of p_h around p_i clockwise. Those lists are now effectively ordered according to the orientation of the edge that ends at p_i, and effectively their easiness to "turn right", and hence preserve convexity. All the $O(kn^{4k-2})$ lists will be treated iteratively in the same manner. Let consider one of those list and name it $\mathcal{L}$. At this point, the weight of each partial convex path on $\mathcal{L}$ is the largest that ends precisely

with $p_h p_i$ (and the other $4k-2$ points associated to $\mathcal{L}$), we now update those weights so that they represent the size of the largest partial convex path that ends with $p_h p_i$ (and the given $4k-2$ points associated to $\mathcal{L}$) or a harder to turn right edge than $p_h p_i$. To do so, iteratively we update the weight of each element in the following manner: If the weight of the element is lower than the weight of the element directly before it, we set the weight of this element to the weight of the element directly before it. After this, the weight of each element is effectively the max of its own weight and the weight of all the elements before it. Now, for all p_j, $j > i$, taken iteratively in their counter-clockwise sorted order we look for the largest partial convex path that ends with $p_h p_i$ (and the $4k-2$ other points) in $\mathcal{L}$, starting from the end. We then replace the edge $p_h p_i$ with the edge $p_i p_j$ (if $p_i p_j$ is compatible with the associated lower edge), and add the weight associated to the addition of this edge, and update the largest partial convex path that ends with $p_i p_j$ (and the $4k-2$ associated points). When moving forward to the next p_j, we do not have to start from the back of $\mathcal{L}$ all over again, we can simply continue from where we were with the previous point thanks to Lemma 17.

Complexity analysis

We first compute the compatible pair of edges by iterating on all of the $O(n^4)$ pairs of edges, each verification takes $O(1)$ time, but requires $O(n^2 \log r)$ preprocessing. We then iterate on $O(n)$ points p_i. For each of those points we sort two lists in $O(n \log n)$ time (Note that all angular sorting could be obtained in $O(n^2)$ time instead of the $O(n^2 \log n)$ time described in this algorithm, but

this does not change the global complexity of the algorithm), we then fetch $O(n^{4k-1})$ elements that we sort into $O(n^{4k-2})$ lists each of size $O(n)$ (as all the sorting is done according to the previously computed angular sorting, this step takes only $O(n^{4k-1})$ time), then for each of those $O(n^{4k-2})$ lists we update the weight and do the appending to the p_j, $j > i$ in $O(n)$ time. Hence the following theorem:

Theorem 18. *The k-potato peeling problem can be solved in $O(n^{4k} + n^2 \log r)$ time where r is the diameter of S, and n its cardinality.*

Chapter Five

Conclusion

In this dissertation we investigated algorithms for finding convexity in digital sets. We investigated two problems related to this topic and presented the first generic linear time algorithm for digital convex set recognition and solving the optimal digital convex polygon problem in 2 dimensions. The approach we use relies on two properties.

The first one is the quickhull algorithm and the fact that, in 2 dimensions, quickhull runs in linear time relative to the number of points for digital convex sets. A question worthy of investigation that naturally arises from this question is the complexity of the 3 dimensional quickhull algorithm on digital convex sets. While known bounds on the number f of faces of any digital convex set S in 3 dimensions, $f = O(V^{1/2})$, where V is the volume of the convex hull S [1] might help proving that the worst case scenario for the quickhull algorithm for digital convex set in 3 dimension is not $O(n^2)$, this approach seems unable to prove, nor disprove, that the time complexity upper bound for 3d digital convex sets might be lowered to $O(n)$.

The second property used, allowed us to reduce the edge minimization

problem to a polygonal separation problem with a linear number of points relative to the size of the input. This property is tightly tied to Diophantine lines and Pick's theorem, as a consequence the lack of result similar to the Euclidean algorithm in 3 dimension makes a similar approach unlikely to work in higher dimensions.

We then changed the topic from finding smallest enclosing convex polygons to finding the largest included ones. Our investigation of this topic resulted in us presenting the first polynomial time algorithm running in roughly $O(n^3)$ time for the digital potato peeling problem, and the first polynomial time algorithm for the digital k-potato peeling problem (when k is fixed). The approach used to solve the potato peeling is similar to the one used to solve the optimal island problem [7], with an adaptation to test the digital convexity. However, using the same technique of embedding the problem in a DAG, but using another representation of the convex hulls, moving from a fan triangulation approach to a top-bottom hull separation we were able to propose a polynomial time algorithm for the k-potato peeling problem. Note that the approach used to solve the digital k-potato peeling can also be adapted to solve an extension of the optimal island problem where the goal would be to find the largest union of k monochromatic islands instead of simply the largest monochromatic island.

Interestingly, several problems simpler than digital potato peeling have no known algorithms that are quicker than the use of the digital potato peeling algorithms presented in this dissertation. Given a lattice set S, one such problem asks to find the largest digital convex triangle, i.e. a digital convex

subset whose convex hull has only 3 vertices. Despite the apparent simplicity of the problem, it is not trivial to find a better approach than naively testing all triplets of points in S and leads to the same complexity as the potato peeling algorithm.

Another, arguably more interesting example is the recognition of the union of two digital convex sets. While the problem can be solved in roughly $O(n^8)$ time using the digital 2-potato peeling algorithm, it seems unlikely for this method to be optimal. Furthermore, in continuous geometry, the recognition of the union of two convex polygons can be solved in $O(n)$ time [57], while solving the continuous potato peeling problem takes $O(n^7)$ time [15]. While comparing the complexities of the continuous and digital version of the problems is not really relevant due to the different nature of the inputs (a set of vertices representing a polygon in continuous geometry, and a set a lattice points in the digital) the apparent simplicity of the problem in continuous geometry seems to be an indication that a better approach is possible for the digital problem. Moreover, while the continuous approach for the recognition of the union of two convex polygons can not be directly used for the digital version of the problem due to the fact that a clear unique contour is not defined for a lattice set, similar techniques deserve to be investigated.

Beyond the fact that the principle of computational digital geometry that consists in treating problems with digital inputs might lead to better results, investigating the connections between problems in computational geometry and digital geometry, such as covering and packing problems [9, 40] for instance, could lead to new tools for both communities.

REFERENCES

[1] George E Andrews. "A lower bound for the volume of strictly convex bodies with many boundary lattice points". In: *Transactions of the American Mathematical Society* 106.2 (1963), pp. 270–279.

[2] Vladimir Igorevich Arnold. "Statistics of integral convex polygons". In: *Functional Analysis and its Applications* 14 (1980), pp. 79–81. DOI: https://doi.org/10.1007/BF01086547.

[3] E Artin. "Geometric Algebra. Interscience Publ". In: *Inc., New York* (1957).

[4] Imre Bárány and Zoltán Füredi. "On the lattice diameter of a convex polygon". In: *Discrete Mathematics* 241.1-3 (2001), pp. 41–50.

[5] Imre Bárány and János Pach. "On the number of convex lattice polygons". In: *Combinatorics, Probability and Computing* 1.4 (1992), pp. 295–302.

[6] Alexander I Barvinok. "A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed". In: *Mathematics of Operations Research* 19.4 (1994), pp. 769–779.

[7] Crevel Bautista-Santiago et al. "Computing optimal islands". In: *Operations Research Letters* 39.4 (2011), pp. 246–251.

[8] Mark de Berg et al. *Computational Geometry: Algorithms and Applications*. 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[9] Peter Brass, William OJ Moser, and János Pach. *Research problems in discrete geometry*. Springer Science & Business Media, 2006.

[10] Jack E Bresenham. "Algorithm for computer control of a digital plotter". In: *IBM Systems journal* 4.1 (1965), pp. 25–30.

[11] Srecko Brlek et al. "Lyndon + Christoffel = digitally convex". In: *Pattern Recognition* 42.10 (2009). Selected papers from the 14th IAPR International Conference on Discrete Geometry for Computer Imagery 2008, pp. 2239–2246. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2008.11.010. URL: http://www.sciencedirect.com/science/article/pii/S0031320308004706.

[12] Timothy M Chan. "Optimal output-sensitive convex hull algorithms in two and three dimensions". In: *Discrete & Computational Geometry* 16.4 (1996), pp. 361–368.

[13] Timothy M Chan. "Optimal partition trees". In: *Discrete & Computational Geometry* 47.4 (2012), pp. 661–690.

[14] Donald R Chand and Sham S Kapur. "An algorithm for convex polytopes". In: *Journal of the ACM (JACM)* 17.1 (1970), pp. 78–86.

[15] Jyun S. Chang and Chee K. Yap. "A polynomial solution for the potato-peeling problem". In: *Discrete & Computational Geometry* 1.2 (1986), pp. 155–182. ISSN: 1432-0444.

[16] Jean-Marc Chassery. "Discrete convexity: Definition, parametrization, and compatibility with continuous convexity". In: *Computer Vision, Graphics, and Image Processing* 21.3 (1983), pp. 326–344. ISSN: 0734-189X. DOI: https://doi.org/10.1016/S0734-189X(83)80047-4. URL: http://www.sciencedirect.com/science/article/pii/S0734189X83800474.

[17] Jean-Marc Chassery and David Coeurjolly. "Optimal shape and inclusion". In: *Mathematical Morphology: 40 Years On*. Springer, 2005, pp. 229–248.

[18] Bidyut Baran Chaudhuri and Azriel Rosenfeld. "ON THE COMPUTATION OF THE DIGITAL CONVEX HULL AND CIRCULAR HULL OF A DIGITAL REGION". In: *Pattern Recognition* 31.12 (1998), pp. 2007–2016. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(98)00065-X. URL: http://www.sciencedirect.com/science/article/pii/S003132039800065X.

[19] Bernard Chazelle. "Cutting hyperplanes for divide-and-conquer". In: *Discrete & Computational Geometry* 9.2 (1993), pp. 145–158.

[20] Bernard Chazelle. "Lower Bounds on the Complexity of Polytope Range Searching". In: *Journal of the American Mathematical Society* 2.4 (1989), pp. 637–666. ISSN: 08940347, 10886834. URL: http://www.jstor.org/stable/1990891.

[21] Bernard Chazelle and Emo Welzl. "Quasi-optimal range searching in spaces of finite VC-dimension". In: *Discrete & Computational Geometry* 4.5 (1989), pp. 467–489.

[22] David Coeurjolly et al. "An elementary algorithm for digital arc segmentation". In: *Discrete Applied Mathematics* 139.1-3 (2004), pp. 31–50. DOI: 10.1016/j.dam.2003.08.003. URL: https://doi.org/10.1016/j.dam.2003.08.003.

[23] Charles J. Colbourn and R.J. Simpson. "A note on bounds on the minimum area of convex lattice polygons". In: *Bulletin of the Australian Mathematical Society* 45.2 (1992), pp. 237–240. DOI: 10.1017/S0004972700030094.

[24] Loic Crombez, Guilherme Dias da Fonseca, and Yan Gérard. "Peeling Digital Potatoes". In: *CoRR* abs/1812.05410 (2018). arXiv: 1812.05410. URL: http://arxiv.org/abs/1812.05410.

[25] Isabelle Debled-Rennesson, Jean-Luc Rémy, and Jocelyne Rouyer-Degli. "Detection of the discrete convexity of polyominoes". In: *Discrete Applied Mathematics* 125.1 (2003). 9th International Conference on Discrete Geometry for Computer Im agery (DGCI 2000), pp. 115–133. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(02)00227-5. URL: http://www.sciencedirect.com/science/article/pii/S0166218X02002275.

[26] G Lejeune Dirichlet. "Über die Reduction der positiven quadratischen Formen mit drei unbestimmten ganzen Zahlen." In: *Journal für die reine und angewandte Mathematik* 1850.40 (1850), pp. 209–227.

[27] H. Edelsbrunner and F.P. Preparata. "Minimum polygonal separation". In: *Information and Computation* 77.3 (1988), pp. 218–232.

[28] Herbert Edelsbrunner. *Algorithms in combinatorial geometry*. Vol. 10. Springer Science & Business Media, 2012.

[29] Eugene Ehrhart. "Geometrie diophantienne-sur les polyedres rationnels homothetiques an dimensions". In: *Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences* 254.4 (1962), p. 616.

[30] David Eppstein et al. "Finding minimum areak-gons". In: *Discrete & Computational Geometry* 7 (1992), pp. 45–58. DOI: 10.1007/BF02187823.

[31] Jiandong Fang et al. "Digital geometry image analysis for medical diagnosis". In: *proceedings of the 2006 ACM symposium on applied computing*. 2006, pp. 217–221.

[32] Fabien Feschet and Laure Tougne. "On the min DSS problem of closed discrete curves". In: *Electronic Notes in Discrete Mathematics* 12 (2003), pp. 325–336. DOI: 10.1016/S1571-0653(04)00496-2. URL: https://doi.org/10.1016/S1571-0653(04)00496-2.

[33] AR Forrest. "Ii. current developments in the design and production of three-dimensional curved objects-computational geometry". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 321.1545 (1971), pp. 187–195.

[34] Yan Gerard. "About the Decidability of Polyhedral Separability in the Lattice $\mathbb{Z}^d$". In: *Journal of Mathematical Imaging and Vision* 59.1 (2017), pp. 52–68.

[35] Yan Gérard. "Recognition of Digital Polyhedra with a Fixed Number of Faces". In: *Discrete Geometry for Computer Imagery*. Ed. by Nicolas Normand, Jeanpierre Guédon, and Florent Autrusseau. Cham: Springer International Publishing, 2016, pp. 415–426. ISBN: 978-3-319-32360-2.

[36] Ronald L. Graham. "An efficient algorithm for determining the convex hull of a finite planar set". In: *Info. Pro. Lett.* 1 (1972), pp. 132–133.

[37] Jonathan Scott Greenfield. *A Proof for a QuickHull Algorithm*. Tech. rep. Syracuse University, 1990.

[38] Christian Haase, Benjamin Nill, and Andreas Paffenholz. "Lecture notes on lattice polytopes". In: *Preliminary version* (2012).

[39] Stephen P Humphries. "Generation of special linear groups by transvections". In: *Journal of Algebra* 99.2 (1986), pp. 480–495.

[40] Xiaodong Jia et al. “Validation of a digital packing algorithm in predicting powder packing densities”. In: *Powder Technology* 174.1-2 (2007), pp. 10–13.

[41] Chul E. Kim and Azriel Rosenfeld. “Convex Digital Solids”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 4.6 (1982), pp. 612–618. DOI: 10.1109/TPAMI.1982.4767314. URL: https://doi.org/10.1109/TPAMI.1982.4767314.

[42] Chul E. Kim and Azriel Rosenfeld. “Digital Straight Lines and Convexity of Digital Regions”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 4.2 (1982), pp. 149–153. DOI: 10.1109/TPAMI.1982.4767221. URL: https://doi.org/10.1109/TPAMI.1982.4767221.

[43] David G Kirkpatrick and Raimund Seidel. “The ultimate planar convex hull algorithm?” In: *SIAM journal on computing* 15.1 (1986), pp. 287–299.

[44] Kazuo Kishimoto. “Characterizing Digital Convexity and Straightness in Terms of Length and Total Absolute Curvature”. In: *Computer Vision and Image Understanding* 63.2 (1996), pp. 326–333. ISSN: 1077-3142. DOI: https://doi.org/10.1006/cviu.1996.0022. URL: http://www.sciencedirect.com/science/article/pii/S1077314296900223.

[45] Gisela Klette. “A Comparative Discussion of Distance Transforms and Simple Deformations in Image Processing”. In: (2003).

[46] Reinhard Klette and Azriel Rosenfeld. *Digital geometry: Geometric methods for digital picture analysis*. Elsevier, 2004.

[47] Reinhard Klette and Azriel Rosenfeld. *Digital geometry: Geometric methods for digital picture analysis*. Morgan Kaufmann, 2004.

[48] Jacques-Olivier Lachaud. “An alternative definition of digital convexity”. In: 2021.

[49] Fu Liu. “On positivity of Ehrhart polynomials”. In: *Recent trends in algebraic combinatorics*. Springer, 2019, pp. 189–237.

[50] Jiři Matoušek. “Efficient partition trees”. In: *Discrete & Computational Geometry* 8.3 (1992), pp. 315–334.

[51] Jiři Matoušek. “Range searching with efficient hierarchical cuttings”. In: *Discrete & Computational Geometry* 10.2 (1993), pp. 157–182.

[52] H. Minkowski. *Geometrie der Zahlen*. Geometrie der Zahlen vol. 2. B.G. Teubner, 1910. URL: https://books.google.fr/books?id=MusGAAAAYAAJ.

[53] Georg Pick. “Geometrisches zur Zahlenlehre”. In: *Sitzungsberichte des Deutschen Naturwissenschaftlich-Medicinischen Vereines für Böhmen "Lotos" in Prag.* v.47-48 1899-1900 (1899).

[54] Stanley Rabinowitz. “$O(n^3)$ Bounds for the Area of a Convex Lattice n-gon”. In: *Geombinatorics* 2.4 (1993), pp. 85–88.

[55] Christian Ronse. “A Bibliography on Digital and Computational Convexity (1961-1988)”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11.2 (Feb. 1989), pp. 181–190. ISSN: 0162-8828. DOI: 10.1109/34.16713. URL: https://doi.org/10.1109/34.16713.

[56] Alexander Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[57] Thomas Shermer. “On recognizing unions of two convex polygons and related problems”. In: *Pattern Recognition Letters* 14.9 (1993), pp. 737–745.

[58] James Joseph Sylvester. *The Laws of Verse: Or Principles of Versification Exemplified in Metrical Translations, Together with an Annotated Reprint of the Inaugural Presidential Address to the Mathematical and Physical Section of the British Association at Exeter*. Longmans, Green, 1870.

[59] Dan E Willard. “Polygon retrieval”. In: *SIAM Journal on Computing* 11.1 (1982), pp. 149–165.

[60] Andrew Chi-Chih Yao. “A lower bound to finding convex hulls”. In: *Journal of the ACM (JACM)* 28.4 (1981), pp. 780–787.

[61] Joviša Žunić. “Notes on Optimal Convex Lattice Polygons”. In: *Bulletin of the London Mathematical Society* 30.4 (1998), pp. 377–385. DOI: 10.1112/S0024609398004482. eprint: https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/S0024609398004482. URL:

https : / / londmathsoc . onlinelibrary . wiley . com / doi / abs / 10 . 1112 / S0024609398004482.