Journal of Mathematical Imaging and Vision
https://doi.org/10.1007/s10851-024-01212-y

Computation of 2D Continuous Geometric Moments Through Inclusion–Exclusion

Paola Magillo[1] · **Lidija Čomić**[2]

Received: 26 February 2024 / Accepted: 26 August 2024

Abstract
We propose a method for computing continuous (exact) geometric moments on 2D binary images, based on a decomposition into overlapping rectangles and on the inclusion–exclusion principle. The approach assumes that the input is given as a chain code describing the contour of the represented object (black pixels) and reduces the number of pixels where calculations are to be performed, especially when the contours have long axis-parallel sides.

Keywords Binary images · Continuous (exact) geometric moments · Contour chain code

1 Introduction

Geometric moments are used for the construction of image descriptors in image analysis and pattern recognition. A set of quantities, invariant to translation, rotation and scaling, have been defined based on geometric moments of order up to three [13]. Moment computation of (foreground) objects contained in digital images is a well-studied field, both for exact integral-based moments and for approximate summation-based ones (for a review, see [10]).

We propose a method to compute the exact moments of digital objects encoded as chain codes describing their contours. From [4], we take the idea of using overlapping axis-parallel rectangles anchored at the corners of the object shape, whose moments are summed up with positive or negative sign. We extend such idea to continuous moments and to the chain code image format. The approach is especially suitable for images where the contours have long axis-parallel sides and few corners. The contributions of this paper are:

- rules to determine the corners during a scan of the chain code,
- formula to compute the moments by adding the signed contributions of the rectangles anchored at the corners (extending [4] to continuous moments and to local knowledge of the pixel colors),
- comparison with other approaches for binary images given as chain codes.

2 Background Notions

We consider binary (black) objects in the plane, with white background. An image is a raster of $N \times M$ pixels, centered at points with integer coordinates, superimposed on the plane, where (digital) objects are sets of pixels.

2.1 Geometric Moments

For an object O in the continuous world, its geometric moment of order $p+q$ is defined as

$$m_{p,q}(O) = \iint_O x^p y^q \mathrm{d}x\mathrm{d}y \tag{1}$$

This research was partially supported by the Science Fund of the Republic of Serbia, #GRANT No 7632, Project "Mathematical Methods in Image Processing under Uncertainty" - MaMIPU.

✉ Paola Magillo
magillo@dibris.unige.it

Lidija Čomić
comic@uns.ac.rs

[1] Department of Computer Science, Bioengineering, Robotics, and Systems Engineering, University of Genova, Genova, Italy

[2] Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia

Published online: 23 September 2024

For a digital object O, the geometric moment

$$m_{p,q}(O) = \sum_{(i,j)\in O} \int_{j-\frac{1}{2}}^{j+\frac{1}{2}} \int_{i-\frac{1}{2}}^{i+\frac{1}{2}} x^p y^q \mathrm{d}x\mathrm{d}y \tag{2}$$

is usually approximated by the discrete sum $\sum_{(i,j)\in O} i^p j^q$. Although negligible for common images and low-order moments, the approximation error may be arbitrarily large for some specific images [10, 18] and grows quickly with p and q [18].

We consider the computation of continuous (exact) geometric moments of 2D binary objects defined by Formula 2. Most existing approaches, including the one we propose here, are based on decomposing the object into (axis-parallel) rectangles. For a rectangle R with opposite vertices (x_1, y_1) and (x_2, y_2), with $x_1 \leq x_2$ and $y_1 \leq y_2$, the integral in Formula (1) solves into

$$m_{p,q}(R) = \frac{1}{(p+1)(q+1)} \left(x_2^{p+1} - x_1^{p+1}\right) \left(y_2^{q+1} - y_1^{q+1}\right) \tag{3}$$

2.2 Chain Codes

An image may be given in a variety of different formats. The most common ones are a 2D matrix of color values, or as a set of runs. A run is a maximal sequence of pixels with equal color in a row and is encoded by its leftmost pixel plus run length. (For a binary image, it is sufficient to provide the black runs.)

An alternative approach, that is, the one considered here, is known as chain code, or contour code, or Freeman code [11]. This encoding provides just the contour of the object O contained in the image. Only the boundary pixels (black pixels adjacent to a white one) are encoded, by giving one of them plus a sequence of moves (unit steps in one of the eight orthogonal and diagonal directions) to incrementally retrieve all the other boundary pixels. Note that, with chain codes, the color of all the pixels of the image is not immediately available.

The chain code image format has many advantages. First of all, it is intuitive, representing the way in which humans draw a picture (constrained to the eight cardinal and diagonal directions). The contour-based paradigm is used in the so-called turtle graphics for introducing children to programming, but also in the postscript vector image format. Contour coded binary images can be easily rendered by filling the interior region of the contour through a flood fill process. Chain codes provide a very effective way to represent image information: they are compact and non-ambiguous, and many information can be directly computed from the chain code, without recovering all the black pixels explicitly. The chain code is generally more compact than the classic matrix of 0/1 values, for most images.

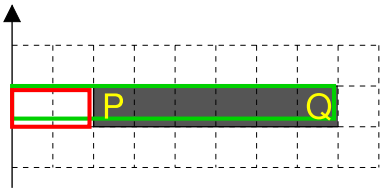

Fig. 1 The two rectangles whose moments define the contribution of a run: the green rectangle with positive sign and the red one with negative sign. The contribution of the red rectangle is subtracted from that of the green rectangle. The boundaries of the two rectangles are slightly perturbed in the image, to avoid overlap

3 Related Work

The algorithms for the computation of the moments of 2D binary images can be classified as decomposition-based and boundary-based [10].

Decomposition-based algorithms decompose the object into simple parts (runs, rectangles or squares) and sum the moments of these parts (if they do not overlap), or apply the inclusion–exclusion principle (if they do).

A natural way to obtain an image decomposition is provided by the popular run-length encoding. Dai et al. [5] compute the contribution of each run as the integral from the left side of the image to the right side of the last pixel in the run minus the integral from the left side of the image to the left side of the first pixel in the run. This approach, illustrated in Fig. 1, extends the delta method of Zakaria et al. [29] to compute the exact continuous moments instead of the approximate discrete ones. Many other variants of this approach have been proposed (e.g., [17]), all based on computing the moments of rows of pixels.

Kerr and Wakenshaw [15] compute the contribution of a run in the same way as in [5], but, like us, consider images encoded through chain codes. While scanning the chain code, they classify the current boundary pixel as starting or ending a run and compute its (positive or negative) contribution to the moment accordingly.

Some methods compute ad-hoc decompositions of the object into non-overlapping rectangles or squares in a preprocessing step. Spiliotis and Mertzios [21, 22] obtain disjoint rectangles by combining consecutive runs of equal spread. Then, they compute the moments of such rectangles by separating the horizontal and vertical summations. The same idea is used by Flusser [8] for exact integration instead of

approximate summation. Sossa and Flusser [1] obtain a similar decomposition through morphological erosion to compute exact moments, while Suk and Flusser [25] use distance transform. Of course, fewer blocks improve the efficiency of moment computation, but usually require more preprocessing time for obtaining the decomposition. An algorithm providing the minimum number of rectangles was given in [6]. Different decomposition schemes were compared in [26], identifying the one in [21, 22] as the better balance between the number of blocks and the decomposition time.

Boundary-based algorithms use some form of the Green theorem to reduce the computational burden from the entire shape to the shape boundary, by transforming the double integral to a non-unique line integral. On a digital image, these approaches ultimately reduce to considering runs [7, 9, 19, 27, 28].

Other algorithms compute the moments of a polygonal region. They either use inclusion–exclusion to decompose the polygon into simpler overlapping ones (triangles and quadrangles) [14, 16, 20, 24], or some form of Green's theorem to transform the double into a line integral, which is computed directly through parametric line equation [2, 23]. Such algorithms are excessively complicated for digital objects, which have only horizontal and vertical edges. Brlek et al. [3] proposed a general framework to process the boundary of a digital image (a sequence of grid edges), summing up a contribution for each edge. This framework can be instantiated to compute the moments.

A common optimization, used by many algorithms, consists of precomputing and storing into tables some quantities, like powers and summations, that are repeatedly used in the moment formulas.

4 The Proposed Method

We assume that the object O contained in the image is the set of black pixels, while white pixels are the background. We pose no restrictions on the configuration of black pixels, so the object does not need to be (simply) connected, or convex. Pixels have integer coordinates, while pixel vertices have half-integer coordinates. The lower-left vertex of the image has coordinates $(-0.5, -0.5)$.

4.1 Outline of the Method

The corner vertices are those vertices V such that the four incident pixels of V are not all white or all black, and are not two edge-adjacent white pixels and two edge-adjacent black pixels. The possible configurations of corner vertices are shown in Fig. 2.

We decompose the object O into overlapping rectangles, having lower-left vertex in $(-0.5, -0.5)$ and upper-right vertex in one of the corners of the object boundary. We compute the moments of each rectangle by using Formula (3) and sum them up with appropriate sign, based on the inclusion–exclusion principle. An example of moment computation with our idea is shown in Fig. 3.

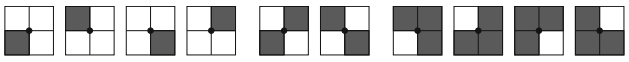

Fig. 2 Configurations of 2×2 pixels defining the corner vertices

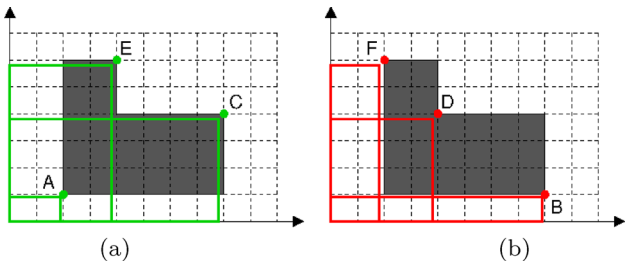

Fig. 3 An object, its six corners A,B,C,D,E,F and the rectangle associated with each of them. **a** The green rectangles give a positive contribution to the moments, **b** the red rectangles give a negative one. The right and upper sides of the rectangles are anchored at the corner, marked with a colored dot (they are not shown at their true position, to avoid overlap)

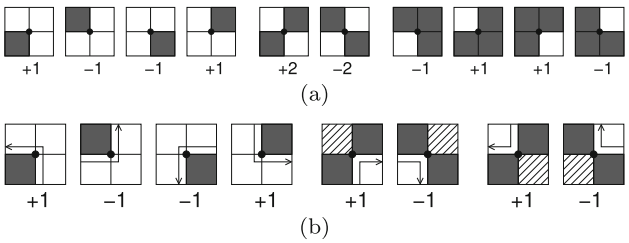

Fig. 4 Coefficients associated with the corner vertices **a** in [4] and **b** here. The color of the dashed pixels is not known

Since all rectangles have the same lower-left corner, the quantities $(z-0.5)^{s+1} - (-0.5)^{s+1}$ involved in Formula (3) can be precomputed and stored for all integer z from 0 up to the maximum between the image height and width, and s up to the maximum order of moments to be computed. Therefore, the computation of the moment of one rectangle needs only one multiplication.

The above idea comes from [4], where it was used to compute discrete moments on an input image given as a raster of black and white pixels. Here, differently from [4], we compute continuous moments instead of discrete ones, and we consider the input object O given as a chain code instead of a binary matrix. We recall that a chain code, also known as Freeman code [11], is an image format providing the object contours as sequences of black pixels, each sequence given as its first (black) pixel, and a sequence of moves.

In [4] the raster image is scanned row by row, keeping two rows in memory (the current one and the previous one).

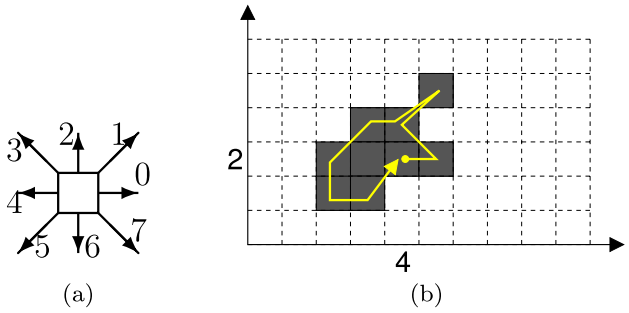

Fig. 5 **a** Numbers encoding the eight directions. **b** An object encoded with the chain code 031545601 and starting pixel (4, 2)

Therefore, when processing a vertex, the configuration of 2×2 pixels around it is completely known. Figure 4a shows the coefficients associated with corner contributions in [4], i.e., the number to be multiplied with the moment of the rectangle having the upper-right vertex in the corner. Note that the cases in which the four pixels sharing the corner form a chessboard configuration counts twice (coefficient $+2$ or -2), because this vertex was examined only once in [4], but two corners occur at it.

Here, we encounter corners during a traversal of chain code describing the boundary of the object. So, a corner in a chessboard configuration will be encountered twice. In addition, since we look at the object "from outside", we have no notion about the color of the third pixel, so we cannot (and we do not need to) distinguish a chessboard configuration from a configuration with three black pixels. Figure 4b shows the coefficients associated with corner contributions in this paper, where the dashed pixel can be of any color.

The correctness of this approach based on overlapping rectangles and the inclusion–exclusion principle follows immediately from the fact that continuous moments (i.e., the integrals defining them) are additive.

4.2 Details of the Method

The input is a chain code describing the contour of a digital object O. The eight orthogonal and diagonal directions in the chain code are represented by the integer numbers in the range $0 \ldots 7$ [11, 15] with the convention shown in Fig. 5a. Starting from the first pixel, and processing the moves in the given order, it is possible to iteratively scan all the (black) pixels of the contour. We assume 8-connected contours, oriented in such a way that the object interior lies to their left. An example of chain code is shown in Fig. 5b.

We assume that a chain code is given by first pixel P_0 with integer coordinates (x_0, y_0) and a sequence of n moves $d_0, d_1, \ldots d_{n-1}$, with integer $d_i \in [0, 7]$ for $0 \leq i \leq n-1$. By starting from P_0 and scanning the list of moves, we get all contour pixels $P_0, P_1, \ldots P_{n-1}$. Each pixel P_i has an incoming move $\text{in}(P_i) = d_{i-1}$ and an outgoing move $\text{out}(P_i) = d_i$ (for P_0, $\text{in}(P_0) = d_{n-1}$).

The configurations where $(\text{in}(P_i) + 5) \bmod 8 = \text{out}(P_i)$ are not possible, because in an 8-connected contour, oriented with the object to its left, P_i would not belong to the contour. We consider all other configurations as possible (note that [15] consider more impossible configurations than us[1]). Based on the pair $\text{in}(P_i)$, $\text{out}(P_i)$ we decide, for each vertex V of P_i, whether V is or is not a corner of the object boundary. All possible pairs $\text{in}(P_i)$, $\text{out}(P_i)$, with the corresponding corners of P_i, are reported in Table 1 and represented in Fig. 6.

In Fig. 6 the dots mark the vertices of P_i that are corners. The ones with white dots must not be counted as corners at P_i, because also the next contour pixel P_{i+1} has a corner in the same vertex, and it must be counted only once. By convention, we count a corner, shared by two contour pixels, only in the second one. The only exception is when $\text{in}(P_i)$, $\text{out}(P_i)$ are opposite diagonal directions (e.g., SW,NE): in that case, the corner V must be counted twice, i.e., at both pixels P_i and P_{i+1}. Therefore, the black dots in Fig. 6 are the corners at P_i, and their directions (relative to the center of P_i) are listed in Table 1.

The vertices of P_i to be considered as corners are established by the following rules, where all additions are modulo 8 (see Table 1). If $\text{in}(P_i)$ is even (i.e., a cardinal direction), the corners are:

- If $\text{out}(P_i)$ is equal to $\text{in}(P_i)$ or $\text{in}(P_i)+7$, then no vertex;
- If $\text{out}(P_i)$ is equal to $\text{in}(P_i)+1$ or $\text{in}(P_i)+2$, then the vertex lying in direction $in(P_i)+7$ (as seen from the center of P_i).
- If $\text{out}(P_i)$ is equal to $\text{in}(P_i)+3$ or $\text{in}(P_i)+4$, then the two vertices lying in directions $\text{in}(P_i)+7$ and $\text{in}(P_i)+1$.
- If $\text{out}(P_i)$ is equal to $\text{in}(P_i)+6$, then the vertex lying in direction $\text{in}(P_i)+6$.

If $\text{in}(P_i)$ is odd (i.e., a diagonal direction), the corners are:

- If $\text{out}(P_i)$ is equal to $\text{in}(P_i)+6$ or $\text{in}(P_i)+7$, then the vertex lying in direction $\text{in}(P_i)+4$,
- If $\text{out}(P_i)$ is equal to $\text{in}(P_i)$ or $\text{in}(P_i)+1$, then the two vertices lying in directions $\text{in}(P_i)+4$ and $\text{in}(P_i)+6$.
- If $\text{out}(P_i)$ is equal to $\text{in}(P_i)+2$ or $\text{in}(P_i)+3$, then the three vertices lying in directions $\text{in}(P_i)+4$, $\text{in}(P_i)+6$, $\text{in}(P_i)$.
- If $\text{out}(P_i)$ is equal to $\text{in}(P_i)+4$, then all four vertices of P_i.

[1] For example, in the first row of Fig. 6 (i.e., $\text{out}(P_i) = 0$) we consider as impossible only $\text{in}(P_i) = 3$, because the current pixel would be completely contained in the object and thus not a boundary pixel. In [15], also $\text{in}(P_i) = 4, 5$ are impossible, i.e., an object cannot have an appendix which is only one pixel wide. For us, such an object is allowed.

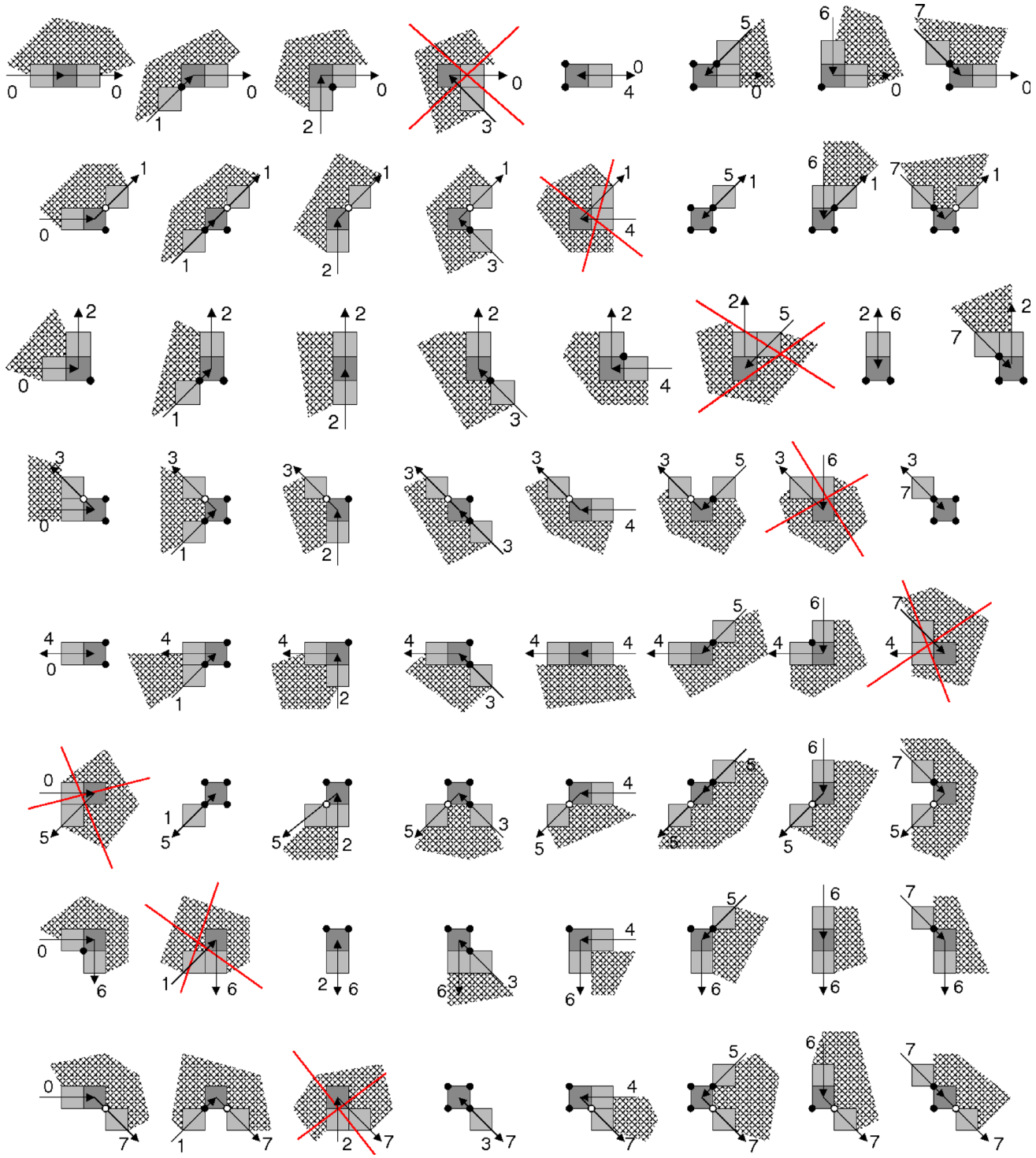

Fig. 6 All configurations for a contour pixel P_i (in darker color), presented in the same order as in Table 1. The interior of the object lies to the left of the contour, and is denoted with a dashed pattern. Impossible configurations are marked with a red cross. The vertices of P_i marked with a dot are corners. In some cases, a corner vertex is shared by two consecutive contour pixels and, by convention, it is counted only in the second pixel. Therefore, only the corner vertices marked with black dots are considered, while those marked with white dots are not counted (they will be counted when processing the next pixel P_{i+1})

Table 1 Vertices of the current pixel P_i, which are counted as corners

		in(P_i)							
		0	1	2	3	4	5	6	7
out(P_i)	0	–	5	7	×	3,5	1,3,5	5	3,5
	1	7	5,7	–	7	×	1,3,5,7	5,7	3,5,7
	2	7	5,7	–	7	1	×	5,7	3,5,7
	3	7,1	5,7,1	1	7,1	–	1	×	3,5,7,1
	4	7,1	5,7,1	1	7,1	–	1	3	×
	5	×	5,7,1,3	1,3	7,1,3	3	1,3	–	3
	6	5	×	1,3	7,1,3	3	1,3	–	3
	7	–	5	×	7,1,3,5	3,5	1,3,5	5	3,5

The vertices of P_i are expressed as directions 1 = NE, 3 = NW, 5 = SW, 7 = SE, relative to the center of P_i. The symbol × marks impossible configurations. The configurations corresponding to each case are represented in Fig. 6

It is easy to verify that the above rules identify all the corners correctly.

The algorithm starts with the initial pixel P_0 and cycles on the moves d_i for $i = 0, 1, \ldots n-1$. At each iteration, it updates the current pixel P_i and determines its corners. If a vertex V of P_i is a corner, it adds to the overall moment the contribution of the rectangle having upper-right vertex in V, multiplied by coefficient 1 or -1 according to the configuration at V (see Fig. 4b). For example, if $\text{in}(P_i) = 1$, $\text{out}(P_i) = 2$, the SW vertex of P_i has coefficient 1 (fifth configuration of Fig. 4b), and its NW vertex has coefficient -1 (second configuration of Fig. 4b).

4.3 Optimizations

Since all our rectangles have one vertex in $(-0.5, -0.5)$ and the other vertex in a point with coordinates $(x-0.5, y-0.5)$ with x, y non-negative integers, Formula (3) reduces to:

$$((x-0.5)^{p+1} - (-0.5)^{p+1})((y-0.5)^{q+1} - (-0.5)^{q+1}) \quad (4)$$

multiplied by the factor $\frac{1}{(p+1)(q+1)}$. This latter is common to all rectangles and, for efficiency, is multiplied only once, at the end of the process.

In order to speed up the computation of rectangle moments, we precompute all values $((z-0.5)^t - (-0.5)^t)$ for all values of integer z from 0 to the maximum image dimension, and of integer t from 0 to the maximum order of the moments to be computed, plus 1. All such values are stored in a matrix T to be accessed later in constant time.

In the first stage, we fill T with the values $(z-0.5)^t$. This requires a number of multiplications linear in the matrix size, by exploiting the fact that

- $(z-0.5)^0 = 1$ for all z;
- for $t \geq 1$ in increasing order, $(z-0.5)^t = (z-0.5)(z-0.5)^{t-1}$ and the second element has been already computed and stored.

In the second stage, we subtract $(-0.5)^{t+1}$ from each entry of T with $z > 0$, finding this value in row $z = 0$ of T, and finally set the entries of the row $z = 0$ of T to $(-0.5)^t - (-0.5)^t = 0$.

With these precomputed values, the second part of Formula (4) is computed by performing just one multiplication, as

$$T[x+1, p+1]\, T[y+1, q+1].$$

Moreover, we note that corners occur along the boundary of the object O in consecutive pairs with equal x-coordinate and opposite coefficients (i.e., the two corners defining a vertical side of O). By collecting the common factor $T[x+1, p+1]$ of these two contributions, we can avoid one multiplication. Therefore, at a corner with new x we just record the value of y, and at the next corner with the same x we add the contribution $T[x+1, p+1]\,(T[y_{\text{curr}}, q+1] - T[y_{\text{prev}}, q+1])$ with its coefficient, where y_{curr}, y_{prev} are the y-coordinate of the current corner and the recorded one of the previous corner, respectively. In order to pair consecutive corners with equal x-coordinate, when more vertices of P_i are corners, it is crucial to process them in the same order in which they are encountered while traversing the boundary counterclockwise. The corners of P_i must be processed by considering first the one lying in the direction equal or immediately following $\text{out}(P_i)$, and then the other ones according to the counterclockwise circular order $1, 3, 5, 7$ = NE, NW, SW, SE. Table 1 gives the corners of P_i in the correct order for processing them.

Input	ours	comp.	ratio
	228	4630	0.05
	280	2112	0.13
	382	1632	0.23
	656	2182	0.30
	1788	5148	0.35
	1316	3296	0.40
	2128	4992	0.43
	1614	3682	0.44
	896	1844	0.49

Input	ours	comp.	ratio
	970	1784	0.54
	1016	1894	0.54
	1472	2646	0.56
	2302	3574	0.64
	968	1384	0.70
	564	760	0.74
	1902	2574	0.74
	3252	4416	0.74
	2008	2406	0.83

Input	ours	comp.	ratio
	1520	1808	0.84
	2860	3182	0.90
	1530	1528	1.00
	4268	4086	1.04
	5922	5428	1.09
	2688	2408	1.12
	3342	2852	1.17
	4138	3534	1.17
	4298	3620	1.19

Input	ours	comp.	ratio
	228	3348	0.07
	382	2670	0.14
	1316	6368	0.21
	1614	6236	0.26
	1016	3154	0.32
	280	832	0.34
	656	1824	0.36
	896	2424	0.37
	1788	4102	0.44

Input	ours	comp.	ratio
	2302	5088	0.45
	970	2126	0.46
	1530	2172	0.70
	1902	2444	0.78
	1472	2064	0.71
	2128	4180	0.51
	564	1024	0.55
	968	1740	0.56
	3252	5240	0.62

Input	ours	comp.	ratio
	2008	3170	0.63
	1520	2356	0.65
	3342	5094	0.66
	4268	4784	0.89
	4138	4310	0.96
	2860	2860	1.0
	5922	5426	1.09
	2688	2380	1.13
	4298	3704	1.16

Fig. 7 The number of nonzero contributions to the moments (number of corners for our method and number of starts/ends of a run for the competitor), with ratio of the former number over the latter one (rounded to the first two decimal digits)

5 Experimental Validation and Discussion

We have designed a method for computing the moments of images given in chain code format; therefore, the competitor methods are to be found in the same category. For this reason, in particular, we do not consider those methods which preprocess a raster image and produce a decomposition into non-overlapping rectangles.

We compare our method with the one by Kerr and Wakenshaw [15], who extended the run-based computation performed in [5] to images encoded by means of chain codes.

The idea, from [5], is that of computing and summing up the moments of each run of the input image. The moment of a run extending from (black) pixel $P = (x, y)$ to (black) pixel $Q = (x + k, y)$ is equal to the moment of the rectangle with leftmost pixel $(0, y)$ and rightmost pixel Q, minus the moment of the rectangle with leftmost pixel $(0, y)$ and rightmost (white) pixel $R = (x - 1, y)$, the immediate left neighbor of P, see Fig. 1.

Also here, the factor $\frac{1}{(p+1)(q+1)}$ can be multiplied once at the end, and powers can be precomputed.

In [15], like in our algorithm, the input is the chain code of the object O contained in the image, and rules are provided to recognize whether the current contour pixel starts or ends a run. We implemented the rules which, given the pair of moves $\text{in}(P_i)$, $\text{out}(P_i)$, decide whether the current pixel P_i of the boundary is the start of a run (i.e., a run starts at the left edge of P_i), and whether P_i is the end of a run (i.e., a run ends at the right edge of P_i). Such rules, not detailed here, are just slightly different from [15], because we assume contours with the opposite orientation, and we consider more possible configurations.

The algorithms have been implemented in Python. We used 27 test images taken from the repository at `https://www.flaticon.com/free-icons/`, and their rotated versions by 90°. The images are shown in Fig. 7. All images have size 512×512, with the exception of three of them, of size 490×490. On such images, we have computed moments $m_{p,q}$ with $p + q \leq 3$, and we have counted the number of corners (for our algorithm) or the number of starts/ends of runs (for the competitor algorithm) giving a nonzero contribution to the moment. Up to a constant, this gives the number of multiplications needed to compute the rectangle moments (not including the ones performed to fill the table of precomputed values, that are equal in both methods). The results are shown in Fig. 7, which also shows the ratio of our number of corners over the number of pixels starting/ending a run. Running times are shown in Fig. 8.

On 43 test images, our method has fewer contributions to the moments, and its running time is faster, while only on the remaining 11 test images the opposite happens. Our method is more convenient on those images where the foreground objects are bounded mostly by axis-parallel sides

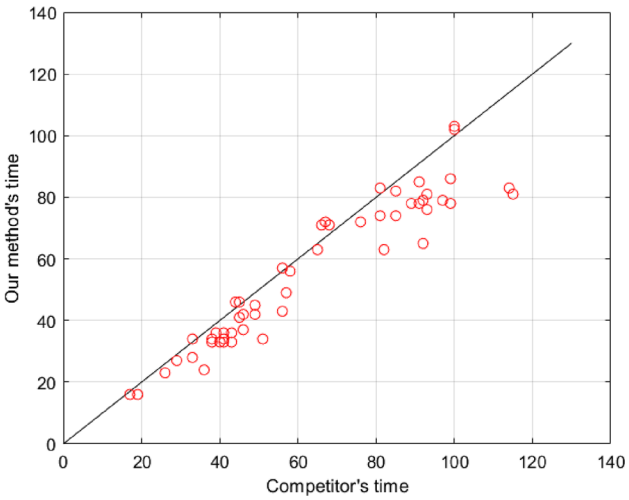

Fig. 8 Comparison of running times of our algorithm (on the vertical axis) and of the competitor one [15] (on the horizontal axis). Each circle is a test image; our method is faster when the circle is below the bisector of the first quadrant (black line)

(long sequences of the same orthogonal direction 0,2,4 or 6, and therefore having few corners). Objects with such characteristics are common products of computer art, such as icons, signals, logos, etc. Conversely, the competitor method is more convenient if the input image has many oblique or curved lines, or many long and thin horizontal lines, which give just one run. In images with axis-parallel lines, the original and the rotated image may have a very different number of runs (e.g., see the suitcase image), while the number of corners is not sensitive to rotation by 90°.

The run-based approach (here represented by [15]) and the corner-based one (proposed by us) are alternatives to the naive scan of the raster image. We also implemented the formula developed by Hosny [12], which sums a contribution to the moment for each black pixel of the given image. The number of contributions in this approach is one or two orders of magnitude larger than the number of corners or of runs.

Brlek et al. [3] proposed a general framework for processing digital objects represented as chain codes. They consider the contour encoded as an inter-pixel line with moves just in the four cardinal directions. The framework has three versions, which compute and add a contribution, respectively: for each horizontal move (i.e., directions E or W), or for each vertical move (i.e., directions N or S), or for each (horizontal or vertical) move. If we apply the second version to moment computation, reformulating it for a contour encoded as a sequence of (black) pixels with moves in eight (cardinal and diagonal) directions, we obtain the same approach used in [15].

6 Concluding Remarks

We have proposed a method for computing (exact) continuous moments on a 2D binary image represented by means of a chain code. The method defines overlapping rectangles anchored in the corners of the object contour and combines their moments by using positive or negative sign, based on inclusion–exclusion. This idea is similar to the one proposed in [4] for computing (approximate) discrete moments, but here we provide a complete algorithm based on contour traversal and pixel classification, not just a formula (and consider continuous moments instead of discrete ones).

The other popular way, based on inclusion–exclusion, to moment computation considers the runs of an image. This approach was embedded into a scan of the chain code by Kerr and Wakenshaw [15]. Compared to it, our method leads to fewer contributions to be computed and thus fewer arithmetic operations to be performed, on most of the test images.

Our contribution will improve the available collection of processing tools for images represented as chain codes, therefore promoting the diffusion of this interesting image format.

Acknowledgements This research was partially supported by the Science Fund of the Republic of Serbia, #GRANT No 7632, Project "Mathematical Methods in Image Processing under Uncertainty"-MaMIPU.

Author Contributions L.C. developed the theory behind the algorithm. P.M. developed the algorithm and made the experiments. Both authors wrote and revised the manuscript.

Funding Open access funding provided by Università degli Studi di Genova within the CRUI-CARE Agreement.

Data Availability No datasets were generated or analyzed during the current study.

Declarations

Conflict of interest The authors declare no conflict of interest.

References

1. Azuela, J.H.S., Flusser, J.: Refined method for the fast and exact computation of moment invariants. In: Progress in pattern recognition, image analysis and applications, 9th Iberoamerican congress on pattern recognition, CIARP, pp. 487–494 (2004)
2. Bamieh, B., de Figueiredo, R.J.P.: A general moment-invariants/attributed-graph method for three-dimensional object recognition from a single image. IEEE J. Robot. Autom. **2**(1), 31–41 (1986)
3. Brlek, S., Labelle, G., Lacasse, A.: The discrete Green Theorem and some applications in discrete geometry. Theor. Comput. Sci. **346**(2–3), 200–225 (2005)
4. Čomić, L., Magillo, P.: Computation of 2D discrete geometric moments through inclusion–exclusion. In: Pattern recognition and artificial intelligence—3rd international conference, ICPRAI, LNCS, vol. 13296. Springer (2022)
5. Dai, M., Baylou, P., Najim, M.: An efficient algorithm for computation of shape moments from run-length codes or chain codes. Pattern Recogn. **25**(10), 1119–1128 (1992)
6. Ferrari, L., Sankar, P., Sklansky, J.: Minimal rectangular partitions of digitized blobs. Comput. Vision Graph. Image Process. **28**(1), 58–71 (1984)
7. Flusser, J.: Fast calculation of geometric moments of binary images. In: 22nd workshop on pattern recognition and medical computer vision (OAGM), pp. 265–274 (1998)
8. Flusser, J.: Refined moment calculation using image block representation. IEEE Trans. Image Process. **9**(11), 1977–1978 (2000)
9. Flusser, J., Suk, T.: On the Calculation of Image Moments. Tech. Rep. 1946, Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic (1999)
10. Flusser, J., Suk, T., Zitova, B.: 2D and 3D image analysis by moments. Wiley, London (2016)
11. Freeman, H.: On the encoding of arbitrary geometric configurations. IRE Trans. Electronic Comput. **10**(2), 260–268 (1961)
12. Hosny, K.M.: Exact and fast computation of geometric moments for gray level images. Appl. Math. Comput. **189**(2), 1214–1222 (2007)
13. Hu, M.-K.: Visual pattern recognition by moment invariants. IRE Trans. Inform. Theory **8**(2), 179–187 (1962)
14. Jiang, X., Bunke, H.: Simple and fast computation of moments. Pattern Recogn. **24**(8), 801–806 (1991)
15. Kerr, D., Wakenshaw, J.T.: Chain code technique for the classification and orientation of simple objects in a machine vision task. In: SPIE conference on machine vision systems for inspection and metroloqy VIII, vol. 3836, pp. 41–51 (1999)
16. Leu, J.-G.: Computing a shape's moments from its boundary. Pattern Recogn. **24**(10), 949–957 (1991)
17. Li, B.C.: A new computation of geometric moments. Pattern Recogn. **26**(1), 109–113 (1993)
18. Liao, S.X., Pawlak, M.: On image analysis by moments. IEEE Trans. Pattern Anal. Mach. Intell. **18**(3), 254–266 (1996)
19. Philips, W.: A new fast algorithm for moment computation. Pattern Recogn. **26**(11), 1619–1621 (1993)
20. Singer, M.H.: A general approach to moment calculation for polygons and line segments. Pattern Recogn. **26**(7), 1019–1028 (1993)
21. Spiliotis, I.M., Mertzios, B.G.: Real-time computation of 2-d moments on block represented binary images on the scan line array processor. In: 8th European signal processing conference, EUSIPCO, pp. 1–4 (1996)
22. Spiliotis, I.M., Mertzios, B.G.: Real-time computation of two-dimensional moments on binary images using image block representation. IEEE Trans. Image Processing **7**(11), 1609–1615 (1998)

23. Steger, C.: On the calculation of arbitrary moments of polygons. Tech. Rep. FGBV-96-05, Forschungsgruppe Bildverstehen (FG BV), Informatik IX Technische Universitaet Muenchen (1996)
24. Strachan, N.J.C., Nesvadba, P., Allen, A.R.: A method for working out the moments of a polygon using an integration technique. Pattern Recogn. Lett. **11**(5), 351–354 (1990)
25. Suk, T., Flusser, J.: Refined morphological methods of moment computation. In: 20th international conference on pattern recognition, ICPR, pp. 966–970 (2010)
26. Suk, T., Hoschl, C., Flusser, J.: Decomposition of binary images—a survey and comparison. Pattern Recogn. **45**(12), 4279–4291 (2012)
27. Tang, G.Y.: A discrete version of Green's Theorem. IEEE Trans. Pattern Anal. Mach. Intell. **4**(3), 242–249 (1982)
28. Yang, L., Albregtsen, F.: Fast computation of invariant geometric moments: a new method giving correct results. In: 12th IAPR international conference on pattern recognition, conference a: computer vision & image processing, ICPR, vol. 1, pp. 201–204 (1994)
29. Zakaria, M.F., Vroomen, L.J., Zsombor-Murray, P.J., van Kessel, J.M.H.M.: Fast algorithm for the computation of moment invariants. Pattern Recogn. **20**(6), 639–643 (1987)

Paola Magillo is an associate professor at the Department of Informatics, Bioengineering, Robotics and Systems Engineering (DIBRIS) of the University of Genova, Italy. Her research interests include computational geometry, geometric modeling, computer graphics, image analysis and processing.

Lidija Čomić is with the Faculty of Technical Sciences, University of Novi Sad, Serbia. Her research interests include digital and computational geometry and topology and image analysis and processing.