Computation of 2D Discrete Geometric Moments through Inclusion-Exclusion

Lidija Čomić[1], Paola Magillo[2]
1. Faculty of Technical Sciences, University of Novi Sad, Novi Sad, Serbia
comic@uns.ac.rs
2. DIBRIS, University of Genova, Genova, Italy
magillo@dibris.unige.it

POST PRINT

This paper has been published on *Lecture Notes in Computer Science N.13363, Pattern Recognition and Artificial Intelligence – Third International Conference, ICPRAI 2022 – Paris, France, June 1–3, 2022 – Proceedings, Part I*
https://link.springer.com/chapter/10.1007/978-3-031-09037-0_43/

Abstract

We propose a new formula for computing discrete geometric moments on 2D binary images. The new formula is based on the inclusion-exclusion principle, and is especially tailored for images coming from computer art, characterized by a prevalence of horizontal and vertical lines. On the target class of images, our formula reduces the number of pixels where calculations are to be performed.

1 Introduction

Geometric moments are a classic tool in image processing and pattern recognition. Based on geometric moments of order up to three, Hu [4] introduced a set of quantities, invariant to similarity transformations (translation, scaling and rotation), through which numerous shape descriptors have been defined. Many formulas for moments computation have been proposed (for a review, see [3]). We propose another one, based on inclusion-exclusion, i.e., on the sum of signed moments of overlapping axis-aligned rectangles. Our formula is especially suitable for images where the objects have a prevalence of vertical and horizontal lines in their contour. This is the case of most computer-produced art, including icons, signals, logos, etc.

2 Background Notions

We consider a bi-dimensional world having two colors, black and white, where the object of interest is black, and the background is white. This world can be continuous or digital. In the latter case, the world is an image, i.e., a raster of $N \times M$ pixels, each either black or white.

2.1 Geometric (Cartesian) Moments

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

$$m_{p,q}(O) = \int_O x^p y^q dx dy.$$

For a digital object O, the geometric moment is usually approximated by

$$m_{p,q}(O) = \sum_{(i,j)\in O} i^p j^q. \tag{1}$$

When the digital object is a rectangle R composed of pixels centered at points in $[I,J] \times [K,L] \cap \mathbb{N}^2$, with I, J, K, L integers,

$$m_{p,q}(R) = \sum_{i=I}^{J} i^p \sum_{j=K}^{L} j^q = (S_p(J) - S_p(I-1)) \cdot (S_q(L) - S_q(K-1)), \tag{2}$$

where S_p and S_q denote the sum of exponentials, defined as $S_k(n) = \sum_{h=1}^{n} h^k$. In particular, if $I = K = 1$ then the rectangle $R(J,L)$ is determined by its upper right vertex with coordinates $(J+\frac{1}{2}, L+\frac{1}{2})$ and Formula (2) becomes

$$m_{p,q}(R(J,L)) = \sum_{i=1}^{J} i^p \sum_{j=1}^{L} j^q = S_p(J) \cdot S_q(L). \tag{3}$$

The sums S_k of exponentials, for $k \leq 3$ (which are the relevant values needed for moment invariants), are given by $S_0(n) = n$, $S_1(n) = n(n+1)/2$, $S_2(n) = n^3/3 + n^2/2 + n/6$, $S_3(n) = n^4/4 + n^3/2 + n^2/4$.

The sums S_k of exponentials, for $k \leq 3$ (which are the relevant values needed for moment invariants), are given by $S_0(n) = n$, $S_1(n) = \frac{n(n+1)}{2}$, $S_2(n) = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$, $S_3(n) = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$.

2.2 Green's Theorem

Green's theorem gives a connection between a double integral over a simply connected region and the line integral along the boundary of the region. We review briefly Green's theorem in the continuous case, and its discrete version.

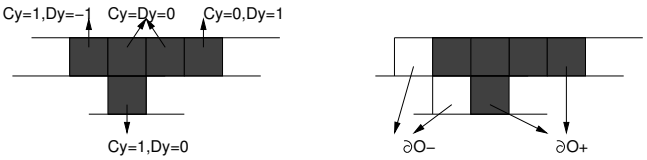


Figure 1: Left: the values of C_Y and D_Y for a run consisting of more pixels (top) or of just one pixel (bottom); only pixels starting or ending a run give a non-zero contribution to the moments. Right: black pixels belonging to ∂O^+ and white pixels belonging to ∂O^- in the same configuration.

If P and Q are continuous functions of two variables with continuous partial derivatives over a simply connected domain O with piece-wise smooth simple closed boundary ∂O, the continuous Green's theorem states that

$$\int_{\partial O} P dx + Q dy = \int_{O} \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx dy,$$

with ∂O oriented counterclockwise. When applied to the computation of moments, Green's theorem is used to convert the double integral to a (non-unique) line integral. One (often used) solution of

$$x^p y^q = \frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y},$$

is $P = \frac{1}{p+1} x^{p+1} y^q$ and $Q = 0$ [14].

The discrete version of Green's theorem, in the formulation proposed by Tang [13], states that

$$\sum_{(i,j)\in O} f(i,j) = \sum_{(i,j)\in \mathcal{C}(O)} \left(F_x(i,j) D_Y(i,j) + f(i,j) C_Y(i,j) \right), \tag{4}$$

where $\mathcal{C}(O)$ is the set of contour pixels of O (i.e., the pixels of O that are edge-adjacent to at least one white pixel), $F_x(i,j) = \sum_{n=0}^{i} f(n,j)$, $D_Y(i,j) = 1$ or -1 if (i,j) is the first or last pixel of a run (a maximal set of contiguous black pixels in one row) with length > 1, otherwise it is 0; $C_Y(i,j) = 1$ if (i,j) is the first pixel of a run, otherwise it is 0 (see Figure 1).

An equivalent formulation of the discrete version of the Green's theorem, proposed by Philips [6], states that

$$\sum_{(i,j)\in O} \nabla_x f(i,j) = \sum_{(i,j)\in \partial O^+} f(i,j) - \sum_{(i,j)\in \partial O^-} f(i,j), \tag{5}$$

where $\nabla_x f(i,j) = f(i,j) - f(i-1,j)$ and ∂O^+ is the set of black pixels with white right neighbor, while ∂O^- is the set of white pixels with black right neighbor (see Figure 1).

3 Related Work

We are interested in the exact computation of discrete moments of digital objects. The relevant algorithms [3] either decompose the object into non-overlapping simple shapes, or they use some form of the discrete Green's theorem.

3.1 Decomposition-Based Algorithms

Algorithms in this class work on a decomposition of the object O, contained in the image, into non-overlapping rectangles. They are either designed for an image encoded in a specific data structure (quadtree [7] or run-length [10, 11, 16]) or they compute a decomposition in a pre-processing step [9, 12].

The δ-method proposed by Zakaria et al. [16] computes low-order moments of horizontally convex objects (i.e., with at most one run in each row). Li [5] generalized this algorithm to non-convex objects [2]. Spiliotis and Mertzios [10, 11] proposed another extension of [16], which first decomposes an object into disjoint rectangular blocks by merging consecutive runs of equal spread and then computes the discrete moments of arbitrary order on the rectangles.

Sossa et al. [9] decompose the object into non-overlapping squares using morphological erosion. Their algorithm works also for objects with holes. Suk and Flusser [12] use distance transform to obtain the decomposition into squares.

The pre-processing stage, necessary to decompose the object, is expensive. Decomposition methods are convenient only if the object is compact (if it can be partitioned into a small number of squares) and a large number of moments is to be computed [12].

3.2 Boundary-Based Algorithms

Tang [13], Philips [6] and Yang and Albregtsen [15] proposed to use the discrete Green's theorem to compute low-order moments. In both formulas coming from the discrete Green's theorem (see Formulas (4) and (5) in Section 2.2), the only pixels giving a non-zero contribution to the sum are those lying on the contour of the object O, i.e, the pixels of O that are edge-adjacent to at least one white pixel.

The algorithm by Tang works on the cyclic sequence $(i_0, j_0), \dots (i_l, j_l)$ of the contour pixels, which is given as a contour chain code. The used formula comes from (4) by taking $f(i, j) = i^p j^q$. This gives

$$F_x(i_n, j_n) = \sum_{h=0}^{i_n} f(h, j_n) = \sum_{h=0}^{i_n} h^p j_n^q = j_n^q S_p(i_n),$$

and

$$
\begin{aligned}
m_{p,q}(O) = & \sum_{(i,j)\in O} f(i,j) = \sum_{(i,j)\in O} i^p j^q \\
= & \sum_{n=0}^{l-1} (F_x(i_n,j_n)D_Y(i_n,j_n) + f(i_n,j_n)C_Y(i_n,j_n)) \\
= & \sum_{n=0}^{l-1} (F_x(i_n,j_n)D_Y(i_n,j_n) + i_n^p j_n^q C_Y(i_n,j_n)).
\end{aligned} \tag{6}
$$

The algorithm by Philips [6] works on the runs, and applies the alternative formulation (5) of the discrete Green's theorem. The algorithm classifies the pixels during a raster scan. For the moment computation, Formula (5) is considered with $f(i,j) = g(i)j^q$, where $g(i)$ is such that $\nabla_x g(i) = i^p$. The moments are computed as

$$
m_{p,q}(\mathrm{O}) = \sum_{(i,j)\in\partial O^+} S_p(i)j^q - \sum_{(i,j)\in\partial O^-} S_p(i)j^q, \tag{7}
$$

where ∂O^+ is the set of end pixels of the runs, and the immediate right neighbors of white pixels in ∂O^- are start pixels of the runs (see Figure 1).

The obtained formula is the same as that of the δ-method of Zakaria et al. [16] (for horizontally convex objects), as formula (5) is equivalent to

$$
\sum_{(i,j)\in O} f(i,j) = \sum_{(i,j)\in\partial O^+} \sum_{h=0}^{i} f(h,j) - \sum_{(i,j)\in\partial O^-} \sum_{h=0}^{i} f(h,j).
$$

The algorithm by Yang and Albregtsen [15] considers the boundary of O (consisting of the edges between black and white pixels) instead of the contour of O (consisting of pixels). For horizontal edges, the term containing C_Y [13] vanishes. For vertical edges, $D_Y = \pm 1$ if the incident black pixel is the end pixel or the start pixel of a run, respectively. The final obtained formula is the same as that by Philips, but in [15] it is embedded within a contour following algorithm.

Flusser [1] and Flusser and Suk [2] improve on the algorithm by Philips by pre-calculating the sums $S_k(n)$, i.e., the integrals over the horizontal runs starting at the left image border. Sossa et al. [8] improved on the algorithm by Philips by computing some moments from runs in the x-direction (if $q > p$), and others from runs in the y-direction (if $q < p$) and simplifying the formulas by expressing them in terms of the pixels in O. Contour pixels are classified through the contour chain code. For objects with holes, the moment contributions of inner contours are subtracted from that of the outer contour.

4 Our Formula

We suppose that the object O is in the first quadrant, and contained in a rectangle of size $N \times M$. That is, O is a set of (black) pixels with integer coordinates (i,j) with $1 \le i \le N$ and $1 \le j \le M$. We pose no restrictions

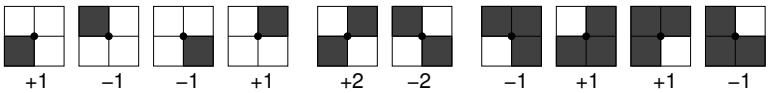


Figure 2: Various configurations defining the corner vertices. The shown coefficient is associated with the lower left pixel of the configuration.

on the configuration of black pixels, so the object does not need to be (simply) connected, or convex.

The coordinates of the four vertices of the pixel (i, j) are $(i \pm \frac{1}{2}, j \pm \frac{1}{2})$. We consider the boundary ∂O as a set of line segments at inter-pixel level. We define the corner vertices of O as 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 configurations of corner vertices are shown in Figure 2.

4.1 The Proposed Formula

We decompose the image into overlapping axis-aligned rectangles. Each rectangle is defined by the vertex $(\frac{1}{2}, \frac{1}{2})$ and by one corner vertex $(J + \frac{1}{2}, L + \frac{1}{2})$ of O. Figure 3 shows an object O and the considered rectangles. According to Formula (3), the moment of the rectangle associated with the corner vertex $(J + \frac{1}{2}, L + \frac{1}{2})$ is equal to $S_p(J) \cdot S_q(L)$.

The moments of O are then computed from rectangle moments through a simple inclusion-exclusion principle. We sum the moments of rectangles for each corner vertex in the boundary of O with the appropriate coefficient, shown in Figure 2. Our formula for moment computation can be summarized as:

$$m_{p,q}(O) = \sum_{(x,y)\in\partial O} V(x, y) \cdot S_p(x - \frac{1}{2}) \cdot S_q(y - \frac{1}{2}) \tag{8}$$

where $V(x, y)$, called corner code, is non-zero for corners only, and its values are shown in Figure 2. Note that $(x - \frac{1}{2}, y - \frac{1}{2})$ are the coordinates of the pixel having (x, y) as its upper right corner.

For the sample object O in the upper part of Figure 3, $m_{0,0}(O) = 8$. We compute $m_{0,0}(O)$ as the sum over all the (non-degenerate) rectangles (see Figure 3, bottom), and therefore we get $+12 - 6 + 4 - 1 + 2 - 8 + 5 = 8$.

4.2 Proof of correctness

Now, we prove that the value $m^R_{p,q}(O)$, computed by our Formula (8) is correct, i.e., equal to $m_{p,q}(O)$, for any object O. The proof is by induction on the number k of pixels in O. For the base case, $k = 0$, O is empty and trivially $m^R_{p,q}(O) = 0 = m_{p,q}(O)$.

We suppose that the formula is correct for objects with up to k pixels, $k \geq 0$, and let O be an object with $k + 1$ pixels. Let $P = (i, j)$ be the rightmost of the

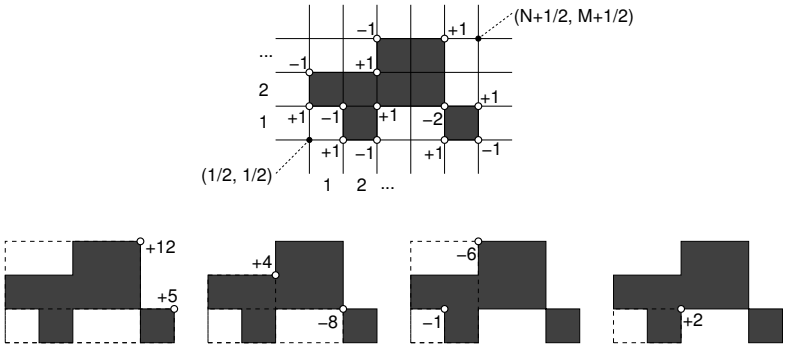
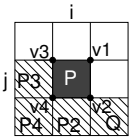


Figure 3: Top: An object O consisting of 8 pixels, the limits N, M, its 13 corner pixels (marked with a white dot) with the coefficient of each. Bottom: the rectangles corresponding to each corner pixel (with the exception of those having $I = 0$ or $J = 0$, which are thus null rectangles), with the values of the associated summands in $m_{0,0}(O)$.



vert.	config.	contribution O	P	O'
v_1		1	1	0
v_3		-1	-1	0
v_3		0	-1	1
v_2		-2	-1	-1
v_2		0	-1	1
v_2		-1	-1	0
v_2		-1	-1	0
...	...	...		

vert.	config.	contribution O	P	O'
v_4		2	1	1
v_4		0	1	-1
v_4		1	1	0
v_4		1	1	0
v_4		1	1	0
v_4		-1	-2	1
v_4		0	1	-1
v_4		0	1	-1

Figure 4: Inductive step of the proof. Left: empty pixels are white, dashed pixels P_2, P_3, P_4 and Q can be black or white. Right: the contribution of each vertex v_1, v_2, v_3, v_4 to the moment of O, of P, and of $O' = O \setminus P$, in all possible configurations.

uppermost pixels in O. This means that the three vertex-adjacent pixels above P, and the one to the right of P are white (see Figure 4).

Let us remove P from O, obtaining O'. By inductive hypothesis, our formula computes correctly the moments of O' and of P. Since moments are additive, we can compute $m_{p,q}(O)$ as $m_{p,q}(O') + m_{p,q}(P)$, i.e., as $m^R_{p,q}(O') + m^R_{p,q}(P)$. We show that this is equal to $m^R_{p,q}(O)$.

Both $m^R_{p,q}(O') + m^R_{p,q}(P)$ and $m^R_{p,q}(O)$ are sums of corner contributions, with the coefficients shown in Figure 2. Differences occur only at vertices v_1, v_2, v_3, v_4 (see Figure 4). Each such vertex v_l is a corner in P, while it may or may not be a corner in O and O'. The table in Figure 4 presents all possible cases, i.e., all possible configurations of pixels incident to v_l, for $l = 1 \ldots 4$, and shows the contribution of v_l to $m^R_{p,q}(O)$, $m^R_{p,q}(P)$, and $m^R_{p,q}(O')$.

As an example, let us consider the upper of the two rows concerning the contribution of the vertex v_3; in such case, v_3 is a corner in O and in P while it is not a corner in O'; its contribution has coefficient -1 in $m^R_{p,q}(O)$ and in $m^R_{p,q}(P)$, and 0 in $m^R_{p,q}(O')$. From the table, we see that the contribution of the vertex v_l to $m^R_{p,q}(O)$ (first numeric column) is always equal to its contribution to $m^R_{p,q}(O') + m^R_{p,q}(P)$ (sum of the last two columns). Therefore $m^R_{p,q}(O') + m^R_{p,q}(P) = m^R_{p,q}(O)$ and thus $m^R_{p,q}(O) = m_{p,q}(O)$.

5 Experimental Validation

The contribution of this paper is in Formula (8). The used formula is just one component of a moment computation algorithm. Other components are the format of the input image and the procedure for pixel scanning. For example, an algorithm may scan a run-length encoded image, or it may follow a contour chain code, etc. Here, we consider those computational costs of an algorithm, which depend on the used formula: the number of pixels giving a non-zero contribution to the moment, and the number of arithmetic operations performed. For comparing our formula with others, we have embedded it into a simple raster scan. The implementation is in C language, using a library for big integers from `https://github.com/dandclark/BigInt`.

In Subsection 5.1, we compare our formula with other representatives of the same class, i.e., boundary-based ones (see Section 3.2). In Subsection 5.2, we compare it with a representative of the decomposition-based class (see Section 3.1).

5.1 Comparison with other boundary-based formulas

We compare our new Formula (8) with two classical formulas based on the discrete Green's theorem, namely Formula (6) used by Tang and Formula (7) used by Philips. We inserted the formulas into a simple image scanning algorithm. Being boundary-based, such formulas could be inserted into an algorithm operating on runs or contours. Our choice was guided by simplicity of implementation.

```
computeMoment(p, q)
1       m = 0 // the moment of order p+q
2       for j = 1 to M
3         a = 0 // the contribution of row j
4         for i = 1 to N
5           c = coefficient(i,j)
6           if (c not zero)
7             a = a + c * ex(i) // add the contribution of pixel (i,j)
8         m = m + a * ey(j) // add contrib. of row j
9       return m // the moment of order p+q
```

Figure 5: General pseudocode of the algorithm used to test the formulas.

All the three considered formulas can be summarized as

$$m_{p,q} = \sum_{i,j} (\text{coefficient}(i,j) \cdot \text{ex}(i) \cdot \text{ey}(j))$$

or equivalently:

$$m_{p,q} = \sum_{j} \left(\text{ey}(j) \cdot \sum_{i} (\text{coefficient}(i,j) \cdot \text{ex}(i)) \right).$$

For our Formula (8), the coefficient is the corner code $V(i+\frac{1}{2}, j+\frac{1}{2})$ (see Section 4 and Figure 2) and $\text{ex}(i) = S_p(i)$, $\text{ey}(j) = S_q(j)$. For Formula (6), the coefficients are actually two, D_y and C_y (see Section 2.2 and Figure 1), $\text{ey}(j) = j^q$, while the inner contribution of the row, i.e., $\text{coefficient}(i,j) \cdot \text{ex}(i)$, consists of two terms $D_y \cdot S_p(i)$ and $C_y \cdot i^p$. For Formula (7), the coefficient is 1 or -1 if (i,j) belongs to ∂O^+ or ∂O^-, respectively (see Section 2.2 and Figure 1), $\text{ex}(i) = S_p(i)$ and $\text{ey}(j) = j^q$.

The general pseudocode of the algorithm used for testing the formulas is shown in Figure 5, where we assumed that the image containing the object has size $N \times M$, column coordinates are $i \in [1, N]$ and row coordinates are $j \in [1, M]$.

All three formulas need the values of $S_k(n)$. Similarly to [1] and [2], all such values are pre-computed and stored in a matrix.

Our formula is especially suited for objects bounded mostly by axis-parallel lines, and therefore having few corners. Objects with such characteristics are common products of computer art, such as icons, signals, logos, etc. We considered a test set consisting of 56 icons from `https://www.flaticon.com/free-icons/` (see Figure 6), and their rotated versions, obtained by exchanging rows and columns. On such 112 test inputs, we computed moments $m_{p,q}$ with $p + q \leq 3$.

We compared the number of pixels contributing to the moment computation (i.e., those with a non-zero coefficient), and the number of performed multiplications and additions (not including the operations needed to compute the sums of exponentials, that are the same for the three formulas). Smaller numbers correspond to more efficient formulas. Formula (6) showed to be less efficient than

Figure 6: Test inputs for the experiments. Each icon was used twice, the second time with a rotation of 90 degrees. Icons are sorted from the best to the worst performance of our formula on them.

Formula (7), therefore in the plots of Figure 7 we compare our Formula (8) with (7) used by Philips. Most dots (precisely, 88 over 112) are above the bisecting line of the first quadrant, meaning that, on the corresponding test inputs, our formula performs better. The ratio of the number of contributing pixels with our formula over the one by Philips ranges from 0.05 to 1.85, with an average value around 0.67. The ratio of the number of operations ranges from 0.05 to 1.68, with an average value around 0.62.

The icons in Figure 6 are sorted from the one giving the best performance (in the top-left corner, where all boundary lines are axis-parallel) to the one giving the worst performance (having no axis-parallel lines). In this last icon, as well as in its rotated version, almost each black pixel is incident to a corner, therefore our formula tests a large number of pixels. Such two input images are the two dots at the extreme right side of the plots in Figure 7.

5.2 Comparison with the decomposition-based approach

We have chosen for the comparison the algorithm by Spiliotis and Mertzios [11], as it is the fastest in its class [3]. We applied the linear-time decomposition algorithm in [11], based on grouping runs having the same extension on consecutive rows. Then, we computed the moment of each block with Formula (2) for rectangles. This second stage is considered in the comparison.

The first plot in Figure 8 compares the number of operations performed for moment computation by [11] and by our formula. As expected, [11] performs fewer operations than us, thanks to the block decomposition built in the preprocessing stage. Nevertheless, our number of operations is only between 1.5 and 2 times larger. The number of corners considered by our formula is about three times the number of blocks in [11], as shown in the second plot if Figure

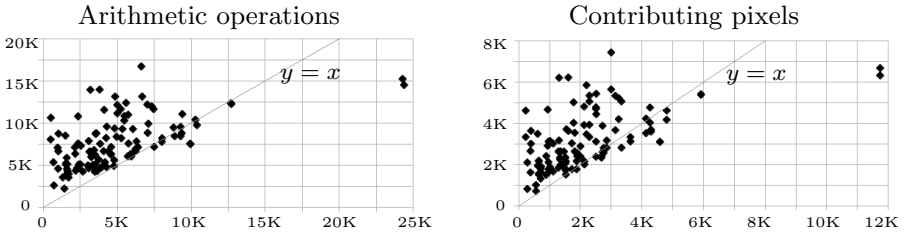


Figure 7: Comparison of Formula (7) by Philips versus our Formula (8). Left: the number of operations (additions and multiplications) in the computation of a single moment. Right: the number of pixels contributing to the computation of moments (pixels with a non-zero coefficient). The horizontal axis is our formula, the vertical one is Formula (7). Each dot is a tested image.

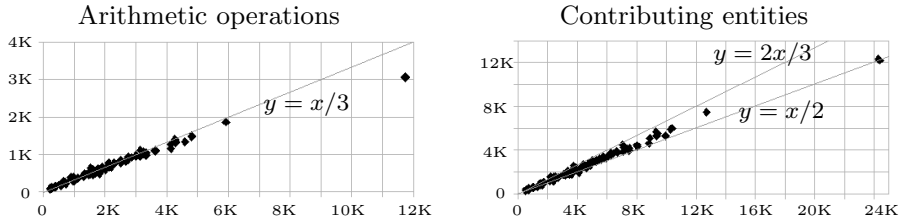


Figure 8: Comparison of Formula (2) applied to the rectangles of the block decomposition by Spiliotis and Mertzios [11] versus our formula applied to the original image. Left: the number of operations. Right: the number of blocks in [11] versus our number of corners. The horizontal axis is our formula, the vertical one is [11]. Each dot is a tested image.

8, with the last image of Figure 6 giving the exceptional ratio of 4.

If, on one hand, our formula doubles the number of necessary operations for moment computation, on the other hand it can directly accept the input image in many common encoding formats (raster, run-length, contour chain), while the one in [11], as well as all decomposition-based ones, needs to build and store the image block representation.

6 Summary and Future Work

We proposed a simple inclusion-exclusion based formula for the computation of discrete geometric moments of 2D binary images, and we showed that it is suitable for images with boundary composed mainly of horizontal and vertical lines. Experiments show that our formula performs fewer arithmetic operations than other boundary-based formulas. Compared with the decomposition-based

approach, it performs a limited amount of extra operations, but methods in this class need to pre-process the given image, while our formula can compute the moments directly. We plan to extend this work to the computation of exact continuous geometric moments in 2D, as well as to images in arbitrary dimensions.

7 Acknowledgments

This research has been partially supported by the Ministry of Education, Science and Technological Development through project no. 451-03-68/2022-14/ 200156 "Innovative scientific and artistic research from the FTS (activity) domain".

References

[1] J. Flusser. Fast Calculation of Geometric Moments of Binary Images. In *22nd Workshop on Pattern Recognition and Medical Computer Vision (OAGM)*, pages 265–274, 1998.

[2] J. Flusser and T. Suk. On the Calculation of Image Moments. Technical Report 1946, Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, 1999.

[3] J. Flusser, T. Suk, and B. Zitova. *2D and 3D Image Analysis by Moments*. John Wiley & Sons, Ltd, 2016.

[4] M.-K. Hu. Visual pattern recognition by moment invariants. *IRE Trans. Information Theory*, 8(2):179–187, 1962.

[5] B. C. Li. A new computation of geometric moments. *Pattern Recognition*, 26(1):109–113, 1993.

[6] W. Philips. A new fast algorithm for moment computation. *Pattern Recognition*, 26(11):1619–1621, 1993.

[7] M. Shneier. Calculations of Geometric Properties Using Quadtrees. *Computer Graphics Image Processing*, 16:296–302, 1981.

[8] J. H. Sossa-Azuela, I. Mazaira-Morales, and J. M Ibarra-Zannatha. An Extension to Philips' Algorithm for Moment Calculation. *Computacion y Sistemas*, 3:5–16, 1 1999.

[9] J. H. Sossa-Azuela, C. Yáñez-Márquez, and J. L. Díaz-de-León S. Computing geometric moments using morphological erosions. *Pattern Recognition*, 34(2):271–276, 2001.

[10] I. M. Spiliotis and B. G. Mertzios. 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*, pages 1–4, 1996.

[11] I. M. Spiliotis and B. G. Mertzios. Real-time computation of two-dimensional moments on binary images using image block representation. *IEEE Trans. Image Processing*, 7(11):1609–1615, 1998.

[12] T. Suk and J. Flusser. Refined Morphological Methods of Moment Computation. In *20th International Conference on Pattern Recognition, ICPR*, pages 966–970, 2010.

[13] G. Y. Tang. A Discrete Version of Green's Theorem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4(3):242–249, 1982.

[14] J. M. Wilf and R. T. Cunningham. Computing region moments from boundary representations. Technical Report NASA-CR-162685), N80-16767, National Aeronautics and Space Administration Jet Propulsion Laboratory California Institute of Technology Pasadena, California, 1979.

[15] L. Yang and F. Albregtsen. 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, Volume 1*, pages 201–204, 1994.

[16] M. F. Zakaria, L. J. Vroomen, P J. Zsombor-Murray, and J. M. H. M. van Kessel. Fast algorithm for the computation of moment invariants. *Pattern Recognition*, 20(6):639–643, 1987.