mathematics

MDPI

Article

On a Fast Hough/Radon Transform as a Compact Summation Scheme over Digital Straight Line Segments

Dmitry Nikolaev [1,2,*], Egor Ershov [1], Alexey Kroshnin [1,3], Elena Limonova [2,4,*], Arseniy Mukovozov [2] and Igor Faradzhev [2,†]

[1] Institute for Information Transmission Problems RAS, 127051 Moscow, Russia; ershov@iitp.ru (E.E.)
[2] Smart Engines Service LLC, 117312 Moscow, Russia
[3] International Laboratory of Stochastic Algorithms and High-Dimensional Inference, Higher School of Economics, 109028 Moscow, Russia; akroshnin@hse.ru (A.K.)
[4] Federal Research Center Computer Science and Control RAS, 119333 Moscow, Russia
* Correspondence: dimonstr@iitp.ru (D.N.); limonova@smartengines.com (E.L.)
† Deceased.

Abstract: The Hough transform, interpreted as the discretization of the Radon transform, is a widely used tool in image processing and machine vision. The primary way to speed it up is to employ the Brady–Yong algorithm. However, the accuracy of the straight line discretization utilized in this algorithm is limited. In this study, we propose a novel algorithm called $ASD2$ that offers fast computation of the Hough transform for images of arbitrary sizes. Our approach adopts a computation scheme similar to the Brady–Yong algorithm but incorporates the best possible line discretization for improved accuracy. By employing the Method of Four Russians, we demonstrate that for an image of size $n \times n$ where $n = 8^q$ and $q \in \mathbb{N}$, the computational complexity of the $ASD2$ algorithm is $O(n^{8/3})$ when summing over $O(n^2)$ digital straight line segments.

Keywords: fast Hough transform; fast discrete Radon transform; digital straight lines

MSC: 65D18

Citation: Nikolaev, D.; Ershov, E.; Kroshnin, A.; Limonova, E.; Mukovozov, A.; Faradzhev, I. On a Fast Hough/Radon Transform as a Compact Summation Scheme over Digital Straight Line Segments. *Mathematics* **2023**, *11*, 3336. https://doi.org/10.3390/math11153336

Academic Editors: Roman Parovik, Kholmat Mahkambaevich Shadimetov and Abdullo Rakhmonovich Hayotov

Received: 3 June 2023
Revised: 25 July 2023
Accepted: 27 July 2023
Published: 29 July 2023

1. Introduction

In this paper, we propose a new fast algorithm for calculating sums along a large number of discrete lines in an image. Such algorithms are called fast Hough transform (fast HT, FHT) or fast discrete Radon transform algorithms [1]. HT was initially positioned as a method for identifying certain graphic primitives in images, such as lines, ellipses, or their segments [2]. Over time, its scope of application has significantly expanded. Some examples of HT application areas that are very far from the search for segments include color segmentation and determination of illumination chromaticity [3–5], as well as automatic determination of optical aberration parameters [6–8].

Similarly to Fourier transform, HT requires fast algorithms for its calculation. The use of fast algorithms is all the more justified the higher the requirements for computational efficiency in the applied problem being solved. Especially stringent performance requirements are imposed on the visual systems of unmanned vehicles. The hardware of such systems can be configured according to the requirements for weight, dimensions, and power consumption, so the computational optimization of the algorithms used on a particular computer becomes extremely important. Therefore, fast algorithms for HT computation have become an almost integral part of such systems. They are most often used to search for road marking elements [9,10], and many authors have created algorithms designed for embedded devices [11–13].

The problem of energy efficiency is also key for systems running on mobile devices [14], which also actively use fast HT calculation. Document recognition systems use HT to search

for quadrangles of document boundaries [15–17] or for parallel-segment patterns. Parallel beam search algorithms using FHT can find the slope of a document [18–20] or its individual characters [21,22] in an image. Another part of the recognition process on mobile devices that uses FHT is barcode detection [23].

For somewhat different reasons, FHT is used in X-ray computed tomography [1,24–26]. With significant computing resources involved, the desired reconstruction resolution parameters have not yet been achieved, so the volumes of processed data are constantly growing, which, when using asymptotically inefficient algorithms, sooner or later leads to a resource shortage. On the contrary, using the correct algorithmic base allows for a much more flexible approach to managing the applied resources. In addition, the task of speeding up reconstruction is extremely relevant for a new class of online methods that allow for real-time evaluation and correction of tomographic measurement parameters [27].

What is interesting is that the use of classical algorithms with FHT does not contradict the development of neural network methods but allows them to be more computationally efficient. A new field in recognition is the construction of so-called Hough networks based on classical convolutional networks. Such networks include an untrainable layer that calculates the discretization of the Radon transform. This layer allows for the extraction of information about lines and segments in an image, even if the receptive fields of convolutional neurons are small, unlike classical convolutional layers [28–30]. This effect is most easily demonstrated by comparing neural network line and segment search methods [31,32]. In recognizing more complex objects with deep learning methods, using FHT also improves the ratio of recognition accuracy and the required amount of computations [33,34]. In addition, currently the main architectures used in neural network tomographic reconstruction methods contain Hough/Radon layers [35–37].

This paper proposes a new Hough transform fast computation algorithm that is more accurate than the Brady–Yong algorithm [1].

2. Related Works

The Hough transform in image processing is usually understood as a method of robust estimation of the parameters of one or more lines in a discrete image by counting the number of points lying on each set of discrete lines. The method is named after Paul V. C. Hough, who first proposed it in 1959 [38] and, in 1962, received, on its basis, the patent, "Method and means for recognizing complex patterns" [39]. It is noteworthy that neither the original publication nor Hough's patent provides a formal definition of the proposed transform. Neither text contains a single mathematical formula. In his historical excursus on the invention of HT, authoritative researcher in classical visual pattern recognition Peter Hart claims [40] that it was Azriel Rosenfeld who first formalized HT, at least partially, in his book on image processing that was published in 1969 [41]. HT (and its very name) gained wide popularity following the publication in of an article in 1972 coauthored by Richard Duda and Peter Hart himself [2].

In 1981, Stanley Deans drew attention [42] to the extreme similarity of the Hough transform and the Radon transform introduced by the latter back in 1917 [43]. In his work, Deans argues that the Radon transform has all the properties of HT (according to R. Duda and P. Hart's interpretation [2]) and proposes the generalization of the Radon transform for curved lines. In their detailed review devoted to HT published in 1988 [44], J. Illingworth and J. Kittler consider, among other things, the work of S. Deans and claim that HT is a special case of the Radon transform, adding that when it comes to binary images, the transforms coincide.

In 1992, Martin Brady and Whanki Yong published a paper on the fast approximate discrete Radon transform [1] for square images of linear size $n \stackrel{\text{def}}{=} 2^q$, $q \in \mathbb{N}$. In its annotation and conclusion, the authors use the names of the Hough and Radon transforms interchangeably. The method they proposed is based on dynamic programming, which is used to skip the repeated calculation of the sums of already processed segments when computing the sum along the line they constitute. This allows for calculation of the HT for

an $n \times n$-sized image in $\Theta(n^2 \log n)$ summations (with the complexity of the naive method $\Theta(n^3)$). Six years later, M. Brady once again published a description of this method, this time without coauthors [45]. Figure 1 explains due to which intersections of patterns (sets of pixel positions) in the discrete case some of the sub-sums participate in the computation of several sums, making it more efficient.

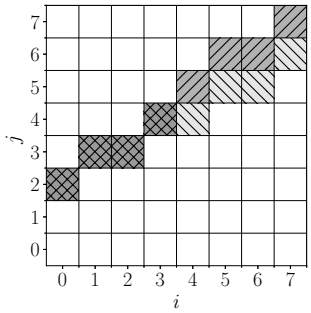

Figure 1. Intersection of two patterns approximating closely spaced straight line segments. The diagonal left shading and lighter gray show the individual part of the pattern with coordinates $j = [4\,i/7 + 2]$, while the right shading and darker gray represent the individual part of the pattern with coordinates $j = [5\,i/7 + 2]$. The checkered area represents the shared part.

The Brady–Yong method was apparently also independently developed by Walter Götz, who published it in his PhD thesis in German in 1993. This work was reprinted in an English-language article by W. Götz and H. Druckmüller, published twice (in 1995 and 1996) in the same journal [46,47]. The two versions of the article are not completely identical, but their differences are, at first glance, quite minor. Another author who published the same method in 1994 is well-known French algorithmist Jean Vuillemin [48]. However, all the works mentioned here were published later than those of M. Brady and W. Yong. Therefore, in this paper, the method is referred to by the names of the latter.

The Brady–Yong method is an approximate method in the sense that the pixel sets it sums over are not the best possible line approximations. In 2021, Simon Karpenko and Egor Ershov published a proof that the maximum orthotropic error of approximating geometric lines in the Brady–Yong method is $(\log_2 n)/6$ for cases of even binary powers (q) of linear image size $n = 2^q$ and $(\log_2 n)/6 - 1/18$ for odd binary powers [49]. Here, the orthotropic error is understood as the error along the y axis for lines defined by the equation $y = k\,x + b$, $|k| \leq 1$ and along the x axis for lines defined by the equation $x = k\,y + b$, $|k| < 1$.

Brady's approach is not the sole method for constructing fast Radon transform algorithms with a computational complexity of $\Omega(n^2 \log n)$. Another approach is the pseudopolar Fourier transform [50]. The algorithm based on this approach is also known as the slant–stack algorithm. Despite having the same asymptotic complexity as the Brady–Yong algorithm, the slant–stack algorithm involves complex multiplications, whereas the Brady–Yong algorithm utilizes only additions and can be easily implemented using integers, thereby allowing for enhanced performance with various SIMD architectures [51]. However, it is important to note that the slant–stack algorithm is not considered to be highly accurate. Levi and Efros [52] showed weight maps defined on the input raster which corresponded to the algorithm's outputs. These weight maps deviated significantly from the expected line discretization. Furthermore, some weights can be negative, resulting in unexpected effects, where the computed Radon image may contain negative values despite having a non-negative preimage. Consequently, ref. [52] proposed an alternative algorithm that is more accurate but slower in terms of execution. Comparing summation-based algorithms directly with algorithms that use soft line discretization poses challenges because the ideal discrete representation of a line depends on the nature of the input data and the specific problem being addressed. In this study, we focus solely on algorithms that define lines in a binary manner. These representations provide a basis for discussing accuracy.

Back in 1974, A. Rosenfeld considered the optimal discretization of line segments, calling its result digital straight line segments (DSLS) [53]. The orthotropic error for DSLS is limited to the constant $1/2$. Thus, even for images with size $n = 16$, the error for discrete Brady–Yong patterns becomes significant compared to DSLS. Figure 2 shows the difference between DSLS and Brady–Yong approximation for a line defined by the equation $j = 6\,i/15$. Figure 3 presents a visualization of the distribution of coordinate error in the Brady–Yong approximation along a line segment for a large image size of $n = 4096$.

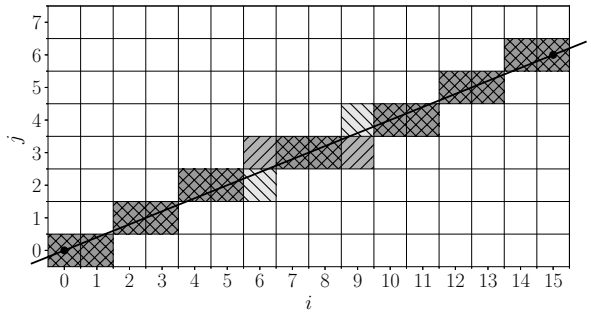

Figure 2. Relative position of the ideal line and its Brady–Yong and DSLS discretizations for $n = 16$, $k = 6/15$. The diagonal left shading and lighter gray represent DSLS discretization, and the diagonal right shading and darker gray represent Brady–Yong discretization.

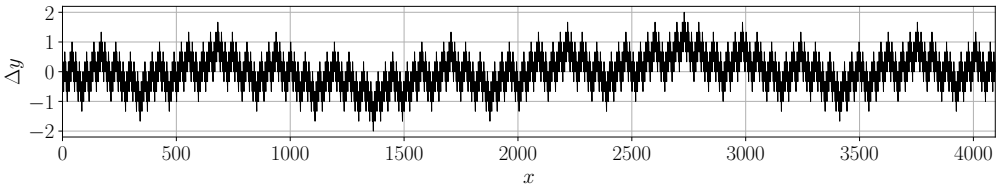

Figure 3. The deviation of Brady–Yong discretization from the ideal line described by the equation $y = x/3$ along the y axis for an image size of $n = 4096$.

Regrettably, there is a lack of research studies that assess the extent to which the mentioned geometric deviations influence the accuracy of applied algorithms utilizing the Brady–Yong method. Nevertheless, this does not imply that the impact is insignificant. Namely, in [25] the authors showed that the acceleration of the tomographic reconstruction method through the convolution and backprojection using the Brady–Yong algorithm results in an approximate doubling of the RMSE of the reconstruction, which is a significant quality loss.

According to these facts, two related questions arise: is it possible to build a faster and/or more accurate FHT algorithm than the algorithm proposed by Brady and Yong? The first question, apparently (the work was not peer-reviewed), was answered by Timur Khanipov in 2018 [54], who showed that the complexity of the problem of computing a dense Hough image using Brady–Yong dyadic patterns is $\Omega(n^2 \log n)$, that is, their method cannot be improved in terms of speed. In the same work, T. Khanipov showed that this estimate is also true for DSLS-based FHT computations. The same year, he proposed, apparently for the first time, a method for accelerated DSLS-based FHT computation [55] with an asymptotic computational complexity of $O(n^3/\log n)$. Just like the Brady–Yong method, the Khanipov method relies solely on summations and achieves acceleration by reusing common sub-sums. The hierarchy of sub-sums is built through an iterative process of refining patterns, where patterns with different slopes are intersected. In essence, while the Brady–Yong method employs a hierarchical division of the image, the Khanipov method employs a hierarchical division of the pattern set based on the patterns' slopes. It is important to note that this method is presented as an abstract mathematical scheme, which means that additional efforts are necessary to turn it into a specific algorithm. It is clear that the difference between the achieved complexity and the previously obtained

estimate is significant and leaves room for more efficient algorithms and/or more accurate lower bounds.

What is interesting to note here is that the algorithm, which is a fast ($\Theta(n^2 \log n)$) and exact discrete Radon transform, has already been constructed [56]. However, this does not negate the problem under discussion. The fact is that this and other works investigating fast algorithms for computation of the discrete Fourier transform have used a different definition of the discrete Radon transform (going back to the 1988 work of I. Gertner [57]). In it, the sets of raster nodes over which the summation is performed are given by linear congruent sequences. For $\mathbb{Z}_n$ residue rings, such sequences are directly analogous to linear functions, but the resulting sets differ significantly from DSLS. In particular, most of them are not connected because for them, k is an integer. Figure 4 illustrates these differences.

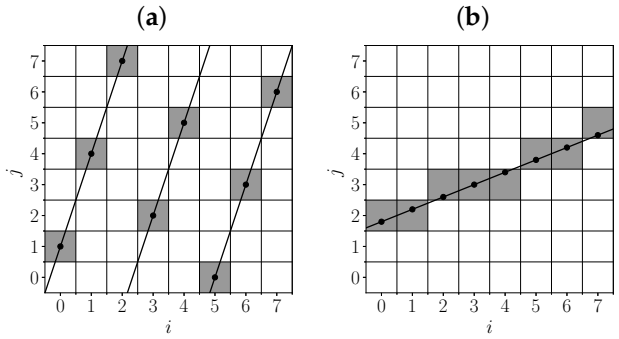

Figure 4. Examples of a set of raster nodes (pixels marked gray), the sum of which is equal to the value of the discrete Radon transform at some point: (**a**) for the transform as proposed by I. Gertner ($k = 3$, $b = 1$) and (**b**) for the Hough transform using DSLS ($k = 0.4$, $b = 1.8$). The linear size of the raster is $n = 8$.

Another approach to the development of computational schemes for FHT is to construct fast generalized Hough transforms (FGHTs) [58]. In this method, the development of the FGHT algorithm (including the FHT for DSLS) is treated as an optimization problem and solved numerically. The disadvantages of this approach include the fact that the scheme has to be calculated separately for each size of input image. In addition, the problem of finding the optimal FGHT generally coincides with the NP-complete problem named "ensemble computation" up to notation (p. 66 [59]). At the same time, approximate approaches with acceptable complexity used for this task are rather primitive (they are based on the greedy strategy [58,60]), and their theoretical efficiency is unknown. As a consequence, estimates of the asymptotic complexity of the resulting FHT computational schemes still have not been obtained.

Thus, in this paper, we solve the problem of constructing a fast algorithm for calculation of the Hough/Radon transform, similar to the Brady–Yong algorithm [1]. Unlike the latter, the proposed algorithm uses a more accurate straight line approximation—DSLS. The closest analog is the Khanipov method [55], but our algorithm has a lower asymptotic complexity.

The remainder of this article is structured as follows. Section 3 introduces the necessary definitions, estimates the cardinality of the set of all DSLS in an image, and considers two algorithms for summing over it: naive $AFNA$, which does not reuse matching sub-sums, and accelerated $AFD2$, which uses a divide-and-conquer scheme. Section 4 specifies the DSLS subset of cardinality ($\Theta(n^2)$) to be calculated and proposes the $AS4R$ algorithm for its computation with $O(n^{8/3})$ summations. Section 5 considers the final $ASD2$ algorithm. We show that the number of summations for it is not worse than for $AS4R$. Finally, the complexity of other operations used in the algorithm is estimated, and the conclusion is drawn that they do not worsen the asymptotic estimate. Section 6 recaps and discusses the obtained results.

3. Hough (Discrete Radon) Transform Algorithms for a Full Set of Digital Straight Line Segments

3.1. Basic Definitions and Statements

In this section, we estimate the total number of DSLS approximating straight line segments intersecting an image section of size $w \times h$, $w, h \in \mathbb{N}$ in such a way that both ends of the segment are located on the section boundary. The image section is understood as the rectangle $\Omega_{w,h} \stackrel{\text{def}}{=} [-0.5, w-0.5) \times [-0.5, h-0.5) \subset \mathbb{R}^2$. The proper image is the mapping $I_{w,h} : \mathbb{Z}^2_{w,h} \to \mathbb{A}$, where $\mathbb{Z}^2_{w,h} \stackrel{\text{def}}{=} \mathbb{Z}_w \times \mathbb{Z}_h$ and $(\mathbb{A}, +)$ is an Abelian semigroup with a neutral element. dom $I_{w,h}$ is called a raster. Raster elements are called positions, and sets (including ordered sets) of the positions are called patterns.

The considered DSLS are expressed in one of two ways depending on the slope of the line being approximated:

$$\begin{aligned} p^H_{w,h}(k,b) &\stackrel{\text{def}}{=} \Big\{ (i,j) \in \mathbb{Z}^2_{w,h} \;\Big|\; j = [k\,i + b] \Big\}, \;\; k \in (-1,1] \subset \mathbb{R}, \;\; b \in \mathbb{R}, \\ p^V_{w,h}(k,b) &\stackrel{\text{def}}{=} \Big\{ (i,j) \in \mathbb{Z}^2_{w,h} \;\Big|\; i = [k\,j + b] \Big\}, \;\; k \in [-1,1) \subset \mathbb{R}, \;\; b \in \mathbb{R}. \end{aligned} \tag{1}$$

The set $(S_{w,h})$ of all considered DSLS is the union of all such patterns and is defined as follows:

$$\begin{aligned} S_{w,h} &\stackrel{\text{def}}{=} S^H_{w,h} \cup S^V_{w,h}, \\ S^H_{w,h} &\stackrel{\text{def}}{=} \Big\{ p \Big| \exists k, b : p = p^H_{w,h}(k,b) \wedge p \neq \varnothing \Big\}, \\ S^V_{w,h} &\stackrel{\text{def}}{=} \Big\{ p \Big| \exists k, b : p = p^V_{w,h}(k,b) \wedge p \neq \varnothing \Big\}. \end{aligned} \tag{2}$$

Patterns $p^H_{w,h}(k,b)$ and $p^V_{h,w}(k,b)$ differ only in the order of coordinates; therefore, we do not consider patterns from S^V further. Additionally,

$$\left|S_{w,h}\right| \leq \left|S^H_{w,h}\right| + \left|S^H_{h,w}\right|. \tag{3}$$

Patterns $p^H_{w,h}(k,b)$ and $p^H_{w,h}(-k,-b)$ are symmetric with respect to the first axis; therefore,

$$\left|S_{w,h}\right| \leq 2\left|S^{H+}_{w,h}\right| + 2\left|S^{H+}_{h,w}\right|, \;\; S^{H+}_{w,h} \stackrel{\text{def}}{=} \Big\{ p \Big| \exists k \geq 0, b : p = p^H_{w,h}(k,b) \wedge p \neq \varnothing \Big\}, \tag{4}$$

and the set $S^{H-} \stackrel{\text{def}}{=} S^H \setminus S^{H+}$ is also not considered.

In what follows, some conclusions are based on the analysis of sets of S patterns closed with respect to the vertical shift:

$$p \in S \implies p + (0,1) \in S, \;\; p + (\Delta i, \Delta j) \stackrel{\text{def}}{=} \{(i + \Delta i, j + \Delta j) | (i,j) \in p\}, \tag{5}$$

but S^{H+} does not have this property.

Let us define patterns for an image that is cyclically closed along the second coordinate:

$$p^{CH+}_{w,h}(k,b) \stackrel{\text{def}}{=} \Big\{ (i,j) \in \mathbb{Z}^2_{w,h} \;\Big|\; j = [k\,i + b] \mod h \Big\}, \;\; k \in [0,1] \subset \mathbb{R}, \;\; b \in \mathbb{R}. \tag{6}$$

The patterns $p^{CH+}_{w,h+w}(k,b)$ and $p^{H+}_{w,h}(k,b)$ coincide on the raster $\mathbb{Z}^2_{w,h}$. Therefore,

$$\left|S_{w,h}\right| \leq 2\left|S^{CH+}_{w,w+h}\right| + 2\left|S^{CH+}_{h,w+h}\right|, \;\; S^{CH+}_{w,h} \stackrel{\text{def}}{=} \Big\{ p \Big| \exists k, b : p = p^{CH+}_{w,h}(k,b) \wedge p \neq \varnothing \Big\}. \tag{7}$$

All patterns from $S^{CH+}_{w,h}$ include exactly one position in each column of the raster $\mathbb{Z}^2_{w,h}$, where the set $S^{CH+}_{w,h}$ has property (5) (with a shift: $\Delta j \in \mathbb{Z}_h$), which allows us to introduce the set of generator patterns ($G^{CH+}_{w,h}$):

$$S^{CH+}_{w,h} = \left\{ p + (0, \Delta j) \middle| p \in G^{CH+}_{w,h} \cap \Delta j \in \mathbb{Z}_h \right\}, \quad G^{CH+}_{w,h} \stackrel{\text{def}}{=} \left\{ p \middle| p \in S^{CH+}_{w,h} \cap (0,0) \in p \right\}. \tag{8}$$

where

$$\left| S^{CH+}_{w,h} \right| = h \left| G^{CH+}_{w,h} \right|, \quad |S_{w,h}| \le 2\,(w+h) \left(\left| G^{CH+}_{w,w+h} \right| + \left| G^{CH+}_{h,w+h} \right| \right). \tag{9}$$

In what follows, we solve the problem of constructing sums of image elements ($I_{w,h}$) given by patterns from $S^{CH+}_{w,h}$. This also solves the problem for $S_{w,h}$, since we have traced not only the relationship between the cardinalities of these sets but also the coincidence of the geometry of their elements up to symmetries.

3.2. Digital Straight Line Segment Count

Let us find an upper estimate for $\left| S^{H+}_{w,h} \right|$. To do this, we prove

Theorem 1. $\left| G^{CH+}_{w,w+h} \right| \le w\, n_1(w) \le w^3$, *where*

$$n_1(w) \stackrel{\text{def}}{=} \frac{(w-2)\,(w-1)}{2} + 1. \tag{10}$$

Proof. Let us estimate the number of patterns from $S^{H+}_{w,h}$ to which the positions $(0, j_l)$ and $(w-1, j_r)$ belong. All such patterns are given by parameter vectors $((k, b))$ lying in the following parallelogram: $b \in [j_l - 0.5, j_l + 0.5)$, $k \in \left[\frac{j_r - b - 0.5}{w-1}, \frac{j_r - b + 0.5}{w-1} \right)$. We denote it as $\omega(j_l, j_r)$. Each of the image columns (i) has no more than two positions that can belong to patterns from the considered set, since

$$\begin{cases} y = k\,x + b, \\ (k, b) \in \omega(j_l, j_r) \end{cases} \quad \Longrightarrow \quad y \in \left[\frac{j_r - j_l}{w-1}\,x + j_l - 0.5, \frac{j_r - j_l}{w-1}\,x + j_l + 0.5 \right), \tag{11}$$

that is, for $(k, b) \in \omega(j_l, j_r)$ and given i, the rounded value in definition (1) lies in a half interval of length 1, and exactly one half-integer value belongs to such a half interval (see Figure 5a).

We denote the half integer belonging to the half interval (11) for column i as t_i. The condition under which $j > t_i$ is satisfied in the position (i, j) specified by definition (1) is linear with respect to the parameters of the following line: $k\,i + b > t_i$; that is, the choice of one of the two possible positions in column i is determined by the belonging of the point (k, b) from $\omega(j_l, j_r)$ to one of the half planes.

With an image width of w, the choice of one of the two possible pixels occurs in $w-2$ or fewer columns, and the pattern is formed by a combination of solutions in these columns. The boundaries of $k\,i + b > t_i$ on the plane of parameters (k, b) of the lines form a partition of the region ($\omega(j_l, j_r)$). The original lines with parameters from $\omega(j_l, j_r)$ correspond to the same pattern from $S^{H+}_{w,h}$ if and only if the corresponding points lie in the same partition region (see Figure 5b–d).

The maximum number of regions that can result from dividing a plane by n straight lines is $n\,(n+1)/2 + 1$, as shown by Jacob Steiner in 1826 [61]. The same is true for a convex region.

Therefore, the function $n_1(w)$ bounds from above the number of patterns with parameters from $\omega(j_l, j_r)$. This is particularly true for $\omega(0, j_r)$, where $\left| G^{CH+}_{w,w+h} \right| \le w\, n_1(w) \le w^3$. □

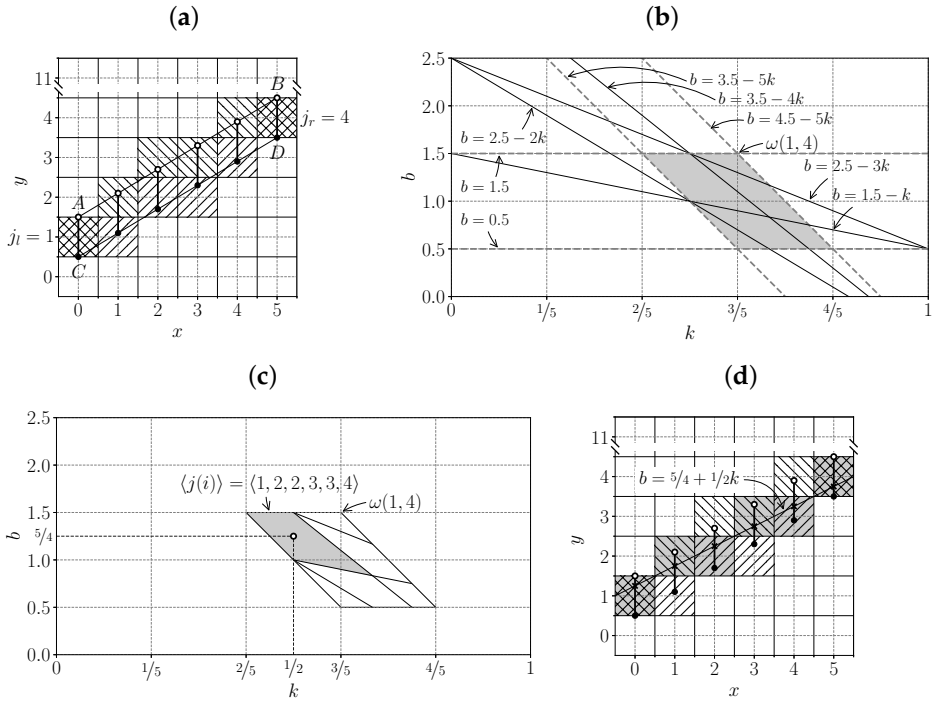

Figure 5. Counting the number of nearest patterns by partitioning the parameter space of lines for a 6×12 raster. (**a**) Lines, the closest pattern for which includes $\{(0, j_l), (w-1, j_r)\}$, lie nowhere above segment AB and everywhere above segment CD; columns $0 < j < w-1$ for such lines have two variants of the nearest position. (**b**) The choice in each column is determined by a linear division of the region $(\omega(j_l, j_r))$ of the considered lines in the parameter space $((k, b))$. (**c**) The parameters of the line $y = 1/2\,x + 5/4$ lie in the region where the nearest pattern for any of the lines is $p = (i, j(i))_{i=0}^{5}$. (**d**) This can be verified by checking whether the line passes above or below the half-integer value in each column.

Corollary 1. $\left|S^{CH+}_{w,w+h}\right| \le w\, n_1(w)\, (w+h)$.

Corollary 2. $\left|S^{H+}_{w,h}\right| \le w\, n_1(w)\, (w+h)$.

Consider Algorithm 1 ($AFNA$ stands for "Algorithm, Full, NAive"), which sums the values of the input image over all patterns from $S^{CH+}_{w,h}$ without taking into account their possible intersections.

The $AFNA$ function takes as input the dimensions of the raster (w and h), as well as the input image (I).

The $Build\_Gchp(w,h)$ function returns a tuple of all patterns from $G^{CH+}_{w,h}$. We do not discuss the method or computational complexity of constructing such a tuple here, as we subsequently eliminate the need for this function.

The $Create\_Zeroed\_Image(w,h)$ function returns an empty $w \times h$ image (i.e., an image with all values equal to the neutral element of $\mathbb{A}$). An image of size $w^3 \times h$ is sufficient to store sums by patterns, since $\left|G^{CH+}_{w,h}\right| \le \left|G^{CH+}_{w,w+h}\right|$, where $\left|S^{CH+}_{w,h}\right| \le w^3\, h$.

Based on the above, the number of summations within semigroup $\mathbb{A}$ in the $AFNA$ algorithm is

$$T_{AFNA}(w,h) \le w^2\, n_1(w)\, h. \tag{12}$$

Up to this point, we have not been taking into account the summation of indices in line 8 of Algorithm 1 and other operations.

Algorithm 1: Naive algorithm $AFNA$ for summation of image values along every pattern from set $S^{CH+}_{w,h}$

Function $AFNA(w,h,I)$ **is**
  Input: $w>0$, $h>0$, image $I:\mathbb{Z}^2_{w,h}\to\mathbb{A}$
  Output: image $J:\mathbb{Z}^2_{w^3,h}\to\mathbb{A}$
  $pl \leftarrow Build\_Gchp(w,h)$; // pl is a tuple of patterns
  $J \leftarrow Create\_Zeroed\_Image(w^3,h)$; // J is an image
  $k \leftarrow 0$;
  foreach $p \in pl$ **do** // p is a pattern, i.e., tuple of positions
    foreach $\langle i,dj\rangle \in p$ **do**
      for $j \leftarrow 0$ **to** $h-1$ **do**
        $J(k,j) \leftarrow J(k,j) + I(i, j+dj \mod h)$;
      end
    end
    $k \leftarrow k+1$;
  end
end

3.3. Divide-and-Conquer Algorithm

Consider Algorithm 2 ($AFD2$ stands for "Algorithm, Full, Dividing by 2"), which sums the values of the input image over all patterns from $S^{CH+}_{w,h}$ and uses the image width bisection method to accomplish this.

Algorithm 2: Divide-and-conquer algorithm $AFD2$ for fast summation of image values along every pattern from set $S^{CH+}_{w,h}$

Function $AFD2(w,h,I)$ **is**
  Input: $w>0$, $h>0$, image $I:\mathbb{Z}^2_{w,h}\to\mathbb{A}$
  Output: image $J:\mathbb{Z}^2_{w^3,h}\to\mathbb{A}$
  if $w>1$ **then**
    $w_L \leftarrow \lfloor w/2 \rfloor$;
    $w_R \leftarrow w - w_L$;
    $I_L \leftarrow Get\_Image\_Window(I,0,0,w_L,h)$; // I_L is an image
    $I_R \leftarrow Get\_Image\_Window(I,w_L,0,w_R,h)$; // I_R is an image
    $J_L \leftarrow AFD2(w_L,h,I_L)$; // J_L is an image
    $J_R \leftarrow AFD2(w_R,h,I_R)$; // J_R is an image
    $pl \leftarrow Build\_Gchp(w,h)$; // pl is a tuple of patterns
    $J \leftarrow Create\_Zeroed\_Image(w^3,h)$; // J is an image
    for $k \leftarrow 0$ **to** $|pl|-1$ **do**
      $p \leftarrow pl(k)$; // p is a pattern, i.e., tuple of positions
      $k_L \leftarrow Get\_Subpattern\_Index(w,h,k,0,w_L)$;
      $k_R \leftarrow Get\_Subpattern\_Index(w,h,k,w_L,w_R)$;
      $pos_R \leftarrow p(w_L)$; // pos_R is a position, i.e., pair of integers
      for $j \leftarrow 0$ **to** $h-1$ **do**
        $J(k,j) \leftarrow J_L(k_L,j) + J_R(k_R, j+pos_R(1) \mod h)$;
      end
    end
  else
    $J \leftarrow I$;
  end
end

The $AFD2$ algorithm uses two new functions.

The $Get\_Image\_Window(I, l, t, w, h)$ function returns an orthotropic rectangular area of the image (I) with the upper-left position (l, t), width (w), and height (h).

The $Get\_Subpattern\_Index(w, h, k, i_0, w_s)$ function parses the pattern with index k from the tuple constructed by the $Build\_Gchp(w, h)$ function. For its subpattern with the first coordinate of positions $i \in [i_0, i_0 + w_s)$, the index of the generator pattern in the tuple is found by the $Build\_Gchp(w_s, h)$ function. Since $Build\_Gchp(w, h)$ constructs all patterns from $G^{CH+}_{w,h}$, the generator pattern can always be found. We do not discuss the method or computational complexity of finding the subpattern index, as we subsequently eliminate the need for this function.

Let us denote the number of summations within semigroup $\mathbb{A}$ in the $AFD2$ algorithm as $T_{AFD2}(w, h)$.

Theorem 2. $T_{AFD2}(w, h) \le \frac{2}{3} w^3 h$.

Proof. Due to Theorem 1, $T_{AFD2}(w, h)$ satisfies the recurrent property:

$$\begin{aligned} &T_{AFD2}(w, h) \le T_{AFD2}(w_L, h) + T_{AFD2}(w_R, h) + w\, n_1(w)\, h, \quad w > 1, \\ &T_{AFD2}(1, h) = 0. \end{aligned} \tag{13}$$

Let us show by induction that $T_{AFD2}(w, h) \le \frac{2}{3} w^3 h$. This holds for $w = 1$ and for $w \ge 2$ by inductive hypothesis:

$$\begin{aligned} \frac{1}{h} T_{AFD2}(w, h) &\le \frac{2}{3}\left(\left\lceil \frac{w}{2} \right\rceil^3 + \left\lfloor \frac{w}{2} \right\rfloor^3\right) + w\, n_1(w) \le \\ &\le \frac{2}{3} \frac{(w-1)^3 + (w+1)^3}{8} + \frac{w^2 - 3w + 4}{2} w = \\ &= \frac{w^3 + 3w}{6} + \frac{w^3 - 3w^2 + 4w}{2} = \\ &= \frac{2}{3} w^3 - \frac{3}{2} w^2 + \frac{5}{2} w \le \frac{2}{3} w^3. \end{aligned} \tag{14}$$

□

4. $O(n^{8/3})$ Summation Algorithm for $O(n^2)$ Digital Straight Line Segments

Both algorithms discussed above ($AFNA$ and $AFD2$) compute sums for all patterns from $S^{CH+}_{w,h}$, among which there are many patterns that barely differ from each other. On the other hand, the Brady–Yong algorithm computes only one pattern for a pair of pixels located at opposite edges of the image. In his work, Brady notes that this number of patterns provides reasonable accuracy [45]. We also consider a variant of the transform in which only one sum is calculated for each pair of pixels at opposite edges of the image. However, unlike the Brady–Yong algorithm, the deviation of the corresponding pattern from the straight line is limited by a constant. We refer to patterns that approximate the lines passing through the centers of pixels at opposite edges of the image as key patterns.

Let us introduce the set ($G^{KCH+}_{w,h}$) of generator key patterns and the set ($S^{KCH+}_{w,h}$) of all key patterns as follows:

$$\begin{aligned} S^{KCH+}_{w,h} &\stackrel{\text{def}}{=} \left\{ p + (0, \Delta j) \,\middle|\, p \in G^{KCH+}_{w,h} \cap \Delta j \in \mathbb{Z}_h \right\}, \\ G^{KCH+}_{w,h} &\stackrel{\text{def}}{=} \left\{ p^{CH+}_{w,h}\left(\frac{k}{w-1}, 0\right) \,\middle|\, k \in [0, w) \subset \mathbb{Z} \right\}. \end{aligned} \tag{15}$$

Obviously,

$$G^{KCH+}_{w,h} \subset G^{CH+}_{w,h}, \quad \left|G^{KCH+}_{w,h}\right| = w, \quad S^{KCH+}_{w,h} \subset S^{CH+}_{w,h}, \quad \left|S^{KCH+}_{w,h}\right| = w\, h. \tag{16}$$

Sparse Transformation and Method of Four Russians

When limiting the number of patterns for which the sums are required as the output, the computational complexity of the naive summation algorithm decreases proportionally to the number of patterns. If summed over the patterns from $S^{KCH+}_{w,h}$, its complexity does not exceed $w^2\,h$, since each pattern consists of w positions.

The restriction on the number of patterns allows for a reduction in the computational complexity of the bisection algorithm if we do not compute partial patterns that do not contribute to any of the final sums. However, it is not easy to estimate the complexity of such an algorithm. We consider an algorithm with lower complexity than the naive algorithm, estimate its complexity, and demonstrate that the bisection algorithm applied to the given set of patterns has a complexity no greater than that of the former algorithm.

We use the adjustment of precalculation and inference complexity, which was first proposed in 1970 by Soviet scientists V. L. Arlazarov, E. A. Dinitz, M. A. Kronrod, and I. A. Faradzhev for fast boolean matrix multiplication and computation of the transitive closure of graphs [62]. The method later became known as the "Method of Four Russians". Fast boolean matrix multiplication is not the only well-known application of this method. For instance, it has recently been utilized for fast summation over arbitrary line segments in images [63], as well as for fast calculation of a sparse discrete John transform in computed tomography [24].

Let us divide the computations into the following two parts:

1. Precalculation: computation of sums for all partial sub-patterns by bisection algorithm up to a certain width (v), which is a parameter of the method;
2. Inference: calculation of the sums for complete patterns by naive addition of the precomputed sums for partial patterns.

Below is Algorithm 3 ($AS4R$ stands for "Algorithm, Sparse, 4 Russians") that follows the scheme presented above and sums the values of the input image over the patterns from $S^{KCH+}_{w,h}$.

The $AS4R$ algorithm uses three new functions.

The $Build\_Gkchp(w,h)$ function returns a tuple of all patterns from $G^{KCH+}_{w,h}$. It is defined below as Algorithm 4.

The $Create\_Empty\_Tuple(n)$ function returns an empty tuple of length n.

The $Get\_Subpattern\_Index\_2(w,h,k,m)$ function parses a pattern with index k from a tuple built with the $Build\_Gkchp(w,h)$ function. For its m-th subpattern, the function searches for the index of the corresponding generator subpattern in the tuple returned by the $Build\_Gkchp$ function during precalculation. We do not discuss either the method or the computational complexity of searching for the subpattern index, since the $AS4R$ algorithm is used solely to estimate the required number of summations.

We denote the number of summations within semigroup $\mathbb{A}$ in the $AS4R$ algorithm as $T_{AS4R}(w,h,v)$.

Theorem 3. $w = 8^q,\ q \in \mathbb{N} \implies T_{AS4R}(w,h,w^{1/3}) \le \frac{5}{3}\,w^{5/3}\,h.$

Proof. We denote the number of summations at the precalculation stage as T_{PRE} and the number at the inference stage as T_{INF}:

$$T_{AS4R}(w,h,v) = T_{PRE}(w,h,v) + T_{INF}(w,h,v). \tag{17}$$

According to Theorem 2, the precalculation complexity is $T_{PRE} \le \frac{2}{3}\,v^3\,h\,w/v$. At the inference stage, for each of the $w\,h$ complete patterns, the precalculated sums over the w/v partial patterns are summed up, that is, $T_{INF} \le w^2\,h/v$.

Therefore, the complexity of the $AS4R$ algorithm is

$$T_{AS4R}(w,h,v) \le \frac{2}{3}\,v^2\,w\,h + \frac{w^2}{v}\,h. \tag{18}$$

With $v = w^{1/3}$, the complexity is

$$T_{AS4R}(w, h, w^{1/3}) \leq \frac{5}{3}\, w^{5/3}\, h. \tag{19}$$

The reasoning leading to this estimate is correct for $w = 8^q$, $q \in \mathbb{N}$. □

Square images with dimensions of $w = h = 8^q$ allow for an even shorter complexity estimate: $O(n^{8/3})$.

Algorithm 3: Four Russians algorithm *AS4R* for fast summation of image values along key patterns from set $S^{KCH+}_{w,h}$

Function $AS4R(w, h, v, I)$ **is**
  Input: $w > 0, h > 0, v > 0$, image $I : \mathbb{Z}^2_{w,h} \to \mathbb{A}$
  Output: image $J : \mathbb{Z}^2_{w,h} \to \mathbb{A}$
  $n \leftarrow \lceil \frac{w}{v} \rceil$;
  $Jl \leftarrow Create\_Empty\_Tuple(n)$; // Jl is a tuple of images
  for $m \leftarrow 0$ **to** $n - 1$ **do**
    $l \leftarrow m\, v$;
    $r \leftarrow \min(l + v, w)$;
    $I_t \leftarrow Get\_Image\_Window(I, l, 0, r - l, h)$; // I_t is an image
    $Jl(m) \leftarrow AFD2(r - l, h, I_t)$;
  end
  $pl \leftarrow Build\_Gkchp(w, h)$; // pl is a tuple of patterns
  $J \leftarrow Create\_Zeroed\_Image(w, h)$; // J is an image
  for $k \leftarrow 0$ **to** $|pl| - 1$ **do**
    $p \leftarrow pl(k)$; // p is a pattern, i.e., tuple of positions
    $J_t \leftarrow Jl(k)$; // J_t is an image
    for $m \leftarrow 0$ **to** $n - 1$ **do**
      $k_t \leftarrow Get\_Subpattern\_Index\_2(w, h, k, m)$;
      $pos_t \leftarrow p(m\, v)$; // pos_t is a position, i.e., pair of integers
      for $j \leftarrow 0$ **to** $h - 1$ **do**
        $J(k, j) \leftarrow J(k, j) + J_t(k_t, j + pos_t(1) \mod h)$;
      end
    end
  end
end

Algorithm 4: *Build_Gkchp* function for the construction of a tuple with elements of set $G^{KCH+}_{w,h}$

Function $Build\_Gkchp(w, h)$ **is**
  Input: $w > 0, h > 0$
  Output: pattern tuple pl
  $pl \leftarrow Create\_Empty\_Tuple(w)$; // pl is a tuple of patterns
  for $k \leftarrow 0$ **to** $w - 1$ **do**
    $p \leftarrow Create\_Empty\_Tuple(w)$; // p is a pattern
    for $i \leftarrow 0$ **to** $w - 1$ **do**
      $p(i) \leftarrow \langle i, \left\lfloor \frac{k\, i}{w-1} \right\rfloor \mod h \rangle$;
    end
    $pl(k) \leftarrow p$;
  end
end

5. Fast Hough (Discrete Radon) Transform Algorithm for $O(n^2)$ Digital Straight Line Segments

5.1. ASD2 Algorithm Description

Let us describe the final version of the algorithm for fast summation of the input image values over all patterns from $S^{KCH+}_{w,h}$. Algorithm 5 (*ASD*2 stands for "Algorithm, Sparse, Dividing by 2") employs the *Build_Gkchp* function introduced above (see Algorithm 4) to build a set of generator patterns, while all the significant operations are encapsulated in the *Calculate_Patterns_ASD*2 function.

Algorithm 5: Divide-and-conquer algorithm *ASD*2 for fast summation of image values along key patterns from set $S^{KCH+}_{w,h}$

Function $ASD2(w, h, I)$ **is**
  Input: $w > 0$, $h > 0$, image $I : \mathbb{Z}^2_{w,h} \to \mathbb{A}$
  Output: pattern tuple pl, image $J : \mathbb{Z}^2_{|pl|,h} \to \mathbb{A}$
  $pl \leftarrow Build\_Gkchp(w, h)$; `// pl is a tuple of patterns`
  $J \leftarrow Calculate\_Patterns\_ASD2(w, h, I, pl)$; `// J is an image`
end

The *Calculate_Patterns_ASD*2 function (see Algorithm 6) takes as input the raster sizes (w, h), the input image (I), and the tuple of generator patterns (pl) to be summed over. The inference algorithm is recursive, bisecting the image along the width.

Algorithm 6: *Calculate_Patterns_ASD*2 function.

Function $Calculate\_Patterns\_ASD2(w, h, I, pl)$ **is**
  Input: $w > 0$, $h > 0$, image $I : \mathbb{Z}^2_{w,h} \to \mathbb{A}$, pattern tuple pl
  Output: image $J : \mathbb{Z}^2_{|pl|,h}$
  if $w > 1$ **then**
    $w_L \leftarrow \lfloor w/2 \rfloor$;
    $w_R \leftarrow w - w_L$;
    $I_L \leftarrow Get\_Image\_Window(I, 0, 0, w_L, h)$; `// I_L is an image`
    $I_R \leftarrow Get\_Image\_Window(I, w_L, 0, w_R, h)$; `// I_R is an image`
    $\langle pl_L, k_L \rangle \leftarrow Get\_Patterns\_Section(pl, 0, w_L)$;
    `/* pl_L is a tuple of patterns, k_L is a tuple of integers */`
    $\langle pl_R, k_R \rangle \leftarrow Get\_Patterns\_Section(pl, w_L, w_R)$;
    `/* pl_R is a tuple of patterns, k_R is a tuple of integers */`
    $J_L \leftarrow Calculate\_Patterns\_ASD2(w_L, h, I_L, pl_L)$; `// J_L is an image`
    $J_R \leftarrow Calculate\_Patterns\_ASD2(w_R, h, I_R, pl_R)$; `// J_R is an image`
    $J \leftarrow Create\_Zeroed\_Image(|pl|, h)$; `// J is an image`
    for $k \leftarrow 0$ **to** $|pl| - 1$ **do**
      $p \leftarrow pl(k)$; `// p is a pattern, i.e., tuple of positions`
      $pos_R \leftarrow p(w_L)$; `// pos_R is a position, i.e., pair of integers`
      for $j \leftarrow 0$ **to** $h - 1$ **do**
        $J(k, j) \leftarrow J_L(k_L(k), j) + J_R(k_R(k), j + pos_R(1) \mod h)$;
      end
    end
  else
    $J \leftarrow I$; `// J is an image`
  end
end

In forward recursion, the image (I) is divided into images I_L and I_R. For the generator patterns from the tuple (pl), fragments belonging to this image are determined, and tuples of subpatterns pl_L and pl_R are formed from them. The corresponding *Get_Patterns_Section* function is not trivial, as we show in the following section. Then, for each image half, the *Calculate_Patterns_ASD*2 function calls itself, passing the corresponding tuple of subpatterns. As a result, images J_L and J_R of the left and right partial sums are formed. These images are required to calculate the sums of J over the full width. The recursion stops at the image width ($w = 1$), when the problem is trivial and the image of the required sums (J) coincides with the input column image (I).

In backward recursion (lines 11–18 of Algorithm 6), for each generator pattern from pl, two generator subpatterns constituting it are determined from the tables of indices (k_L and k_R) returned by the *Get_Patterns_Section* function, and the full sums for all integer vertical pattern shifts are calculated from the images of partial sums (J_L and J_R).

Let us now consider the *Get_Patterns_Section* function (see Algorithm 7).

Algorithm 7: *Get_Patterns_Section* function.

```
1  Function Get_Patterns_Section(pl, i_0, w) is
       Input: pattern tuple pl, i_0 ≥ 0, w > 0, 0 ≤ k < |pl| ⟹ |pl(k)| ≥ i_0 + w
       Output: pattern tuple spl, |spl| ≤ |pl|, index tuple ind, |ind| = |pl|
2      tab ← Create_Empty_Tuple(|pl|);
       /* tab is a tuple of records; each record is a tuple consisting
          of subpattern hash, subpattern itself and index of its parent
          pattern                                                      */
3      for k ← 0 to |pl| − 1 do
4          p ← pl(k) ;            // p is a pattern, i.e., tuple of positions
5          sp ← Create_Empty_Tuple(w) ;                  // sp is a pattern
6          pos_0 ← p(i_0) ;        // pos is a position, i.e., pair of integers
7          for i ← 0 to w − 1 do
8              pos = p(i_0 + i) ;  // pos is a position, i.e., pair of integers
9              sp(i) ← ⟨i, pos(1) − pos_0(1)⟩;
10         end
11         hash ← Get_Pattern_Hash(sp) ;    // hash is a tuple of 4 integers
12         tab(k) ← ⟨hash, sp, k⟩;
13     end
14     Sort_By_Field(tab, 0);
15     spl ← Create_Empty_Tuple(|pl|) ;          // spl is a tuple of patterns
16     ind ← Create_Empty_Tuple(|pl|) ;          // ind is a tuple of integers
17     hash_prev ← ⟨⟩ ;                  // hash_prev is a tuple of integers
18     n ← 0;
19     foreach rec ∈ tab do                        // rec is a record, see above
20         k ← rec(2);
21         hash_cur ← rec(0) ;               // hash_cur is a tuple of 4 integers
22         if hash_cur ≠ hash_prev then
23             spl(n) ← rec(1);
24             n ← n + 1;
25         end
26         ind(k) ← n − 1;
27         hash_prev ← hash_cur;
28     end
29     Shrink_Tuple(spl, n);
30 end
```

The algorithm uses three new functions.

The $Sort\_By\_Field(t, i)$ function sorts a tuple of records (t) according to an element with an index (i) in each record.

The $Shrink\_Tuple(t, n)$ function removes all elements from the tuple (t), except the first n.

The $Get\_Pattern\_Hash(p)$ function builds a unique identifier for the pattern (p) consisting of four integers. The algorithm behind this function is described in [64].

In the first loop (lines 3–13 of Algorithm 7), the *Get_Patterns_Section* function builds a generator subpattern for each input pattern corresponding to a given interval of the first coordinate and calculates its unique identifier. The subpattern tuple is then sorted by identifiers.

In the second loop (lines 19–27), matching subpatterns are excluded from the list. The index table (ind) is supported, allowing each pattern to find its subpattern from the shortened list.

5.2. Algorithm Complexity Analysis

Let us determine the complexity of the $ASD2$ algorithm. First, we estimate the number ($T_{ASD2}(w, h)$) of summations within semigroup $\mathbb{A}$, then examine the asymptotic complexity of auxiliary operations.

Theorem 4. $w = 8^q$, $q \in \mathbb{N} \implies T_{ASD2}(w, h) \leq \frac{5}{3} w^{5/3}\, h$.

Proof. Algorithm $ASD2$ sums the values across subpatterns, doubling their length.

We denote the number of summations performed when constructing subpatterns of length up to and including $v = w^{1/3}$ as T_{SML}, and the number of remaining summations as T_{BIG}:

$$T_{ASD2}(w, h) = T_{SML}(w, h) + T_{BIG}(w, h). \tag{20}$$

$T_{SML}(w, h) \leq T_{PRE}(w, h, w^{1/3})$, since the $AS4R$ algorithm uses the same order of summation but, at the same time, calculates the sums for all patterns, and $ASD2$ only calculates the necessary sums.

In addition, $T_{BIG}(w, h) \leq T_{INF}(w, h, w^{1/3})$, since the $AS4R$ algorithm uses a naive algorithm at the inference stage, and $ASD2$ does not perform unnecessary summations. Indeed, for each pattern out of $G^{KCH+}_{w,h}$, there is not more than $w/w^{1/3}$ summations of precomputed values for subpatterns with lengths not less than $w^{1/3}$, which corresponds to the number of summations at the inference stage of the $AS4R$ algorithm.

Hence, $T_{ASD2}(w, h) \leq T_{AS4R}(w, h, w^{1/3})$. □

We now consider the auxiliary operations.

The $Build\_Gkchp(w, h)$ function has a computational complexity of $O(w^2)$. It is essential that only the key patterns are considered and that the function builds generator elements only.

The number of summations in semigroup $\mathbb{A}$ in the $Calculate\_Patterns\_ASD2$ function, excluding recursion, is $O(|pl|\, h)$. Let us ensure that the other operations satisfy this asymptotic so that the asymptotic complexity of the entire $ASD2$ algorithm is determined by the $T_{ASD2}(w, h)$ number.

The number of index summations in line 16 of Algorithm 6 is equal to the number of summations in semigroup $\mathbb{A}$.

The *Get_Image_Window* function does not involve copying image values and can be performed in $O(1)$ operations.

The $Create\_Zeroed\_Image(w, h)$ function has a computational complexity of $O(w\, h)$; therefore, the $Create\_Zeroed\_Image(|pl|, h)$ call has an acceptable complexity.

Finally, we determine the conditions under which the $Get\_Patterns\_Section(pl, i_0, w)$ function has a computational complexity of $O(|pl|\, h)$.

The $Get\_Pattern\_Hash(p)$ function has a computational complexity of $O(|p|)$ [64], which makes the first loop (lines 3–13 of Algorithm 7) of the *Get_Patterns_Section* function have a computational complexity of $O(|pl|\,w)$.

Sorting in line 14 has a computational complexity of $O(|pl|\,\log|pl|)$. On the other hand, $|pl| = O(w^3)$, which gives $O(|pl|\,\log w)$ operations for the sorting.

The second loop (lines 19–27) of the *Get_Patterns_Section* function has a computational complexity of $O(|pl|)$.

Thus, for $w = O(h)$, the $Get\_Patterns\_Section(pl, i_0, w)$ function has a computational complexity of $O(|pl|\,h)$, and the $ASD2$ algorithm is $O(w^{5/3}\,h)$. For $w = 8^q$, $q \in \mathbb{N}$, the computational complexity of the $ASD2$ algorithm is equal to $O(w^{5/3}\,h)$.

5.3. Properties of the ASD2 Algorithm

It is worth comparing the aforementioned results with the complexity of the Khanipov method. According to Theorem 5.9 in [54], the number of summations (in terms of the current work) for patterns from $S^{KCH+}_{w,h}$ is upper-bounded as follows:

$$\frac{4\,w^2\,h}{1+\log_2 w}\left(1+\sqrt{\frac{2}{w}}\right) \approx \frac{4\,w^2\,h}{\log_2 w}. \tag{21}$$

It should be noted that Theorem 4 guarantees that the complexity bound for the $ASD2$ algorithm is

$$\frac{12\,w^{1/3}(1+\sqrt{2/w})}{5(1+\log_2 w)} > \frac{3}{2} \tag{22}$$

times smaller, and this ratio tends to infinity as $w \to \infty$.

Table 1 presents numerical estimates of computation complexity and geometric deviation for the proposed $ASD2$ algorithm in comparison to previously known methods. The Khanipov method is denoted as KHM, the Brady–Yong algorithm as BYA, and the naive summation of key patterns as $ASNA$. The geometric deviation is evaluated along the y axis for lines described by the following equation: $y = k\,x + b$, $|k| \le 1$. The estimates are calculated for square images of size $n \times n$, $n \in \{256, 1024, 4096\}$. These dimensions are common in the processing of two-dimensional histograms and images and tomographic reconstruction. According to the obtained estimates, for $n = 4096$, the $ASD2$ algorithm exhibits a nearly threefold performance gain compared to the Khanipov method. However, it is important to note that this ratio is for the upper-bound estimates, both of which may be inaccurate.

Table 1. Upper bounds for computation complexity and geometric deviation for different summation-only HT calculation algorithms.

n **Algorithm**	**256**	**1024**	**4096**	**256**	**1024**	**4096**
		Summations			**Error**	
$ASNA$	1.68×10^7	1.08×10^9	6.88×10^{10}	0.5	0.5	0.5
KHM	8.12×10^6	4.08×10^8	2.17×10^{10}	0.5	0.5	0.5
$ASD2$	4.41×10^6	1.78×10^8	7.16×10^9	0.5	0.5	0.5
BYA	5.25×10^5	1.05×10^7	2.02×10^8	1.34	1.67	2

Figure 6 illustrates the impact of geometric approximation errors on the values of the Hough image. The test image is the Shepp–Logan phantom [65], with a resolution of 4096×4096 pixels. When constructing the Hough image using the Brady–Yong algorithm, noticeable high-frequency artifacts are observed. Conversely, these artifacts are absent when employing the DSLS patterns.

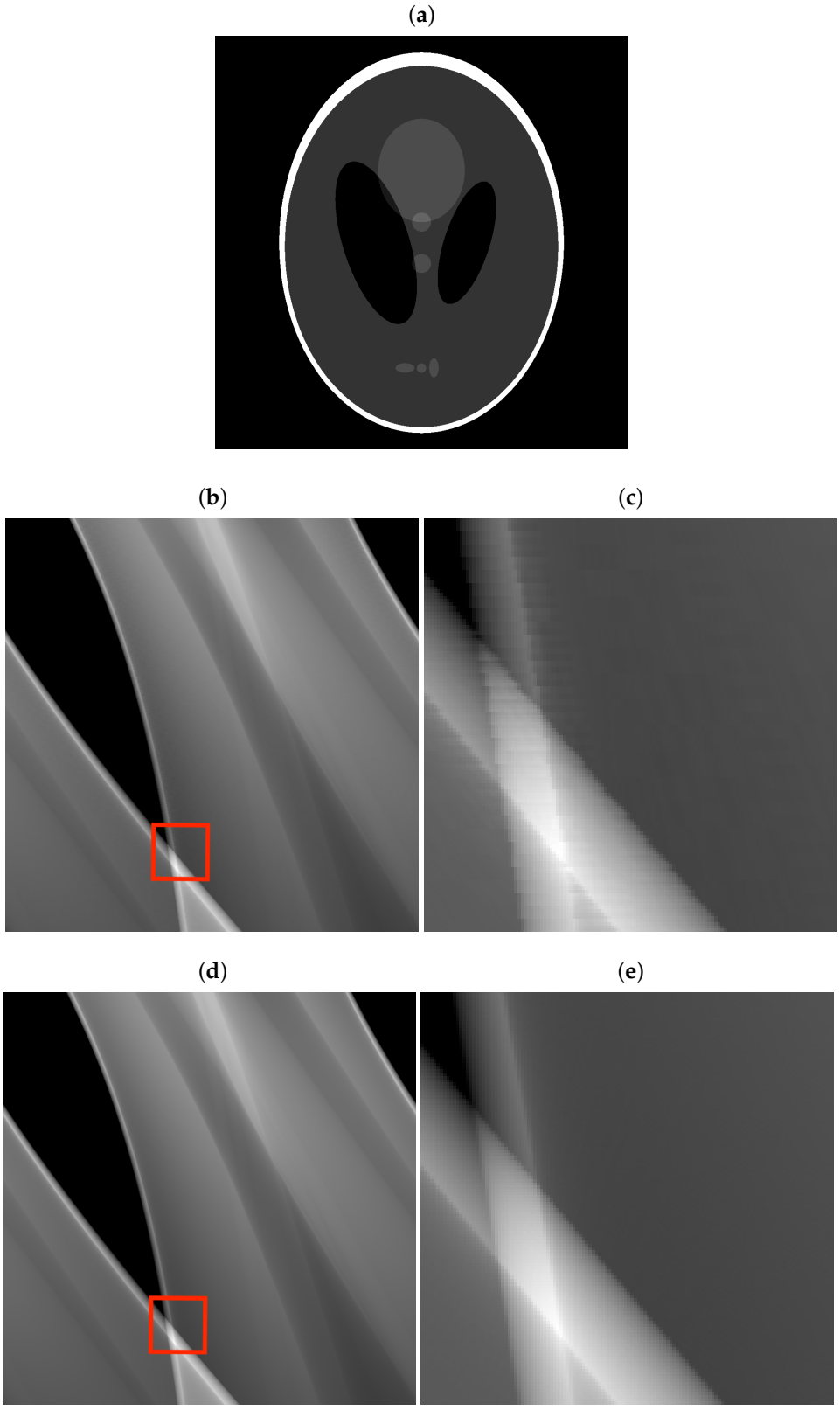

Figure 6. An example of image transformation: (**a**) original image (Shepp–Logan phantom); (**b**) Brady–Yong algorithm; (**c**) Brady–Yong algorithm (a zoomed-in section marked red in the left image); (**d**) $ASD2$ algorithm; (**e**) $ASD2$ algorithm (a zoomed-in section marked red in the left image).

5.4. Final Touches

With the aforementioned constraints, the $ASD2$ algorithm has a complexity of $O(w^{5/3}\,h)$ when processing an image of size $w \times h$. It computes the sums over the patterns from the set $S^{KCH+}_{w,h} \subset S^{CH+}_{w,h}$, while initially, we were faced with the task of summing over patterns from $S_{w,h}$. In Section 4, we decided to limit ourselves to a smaller number of patterns, which we referred to as key patterns.

Let us define the key patterns from the set $S_{w,h}$:

$$\begin{aligned} S^{K}_{w,h} &\stackrel{\text{def}}{=} S^{KH}_{w,h} \cup S^{KV}_{w,h}, \\ S^{KH}_{w,h} &\stackrel{\text{def}}{=} \left\{ p = p^{H}_{w,h}\left(\frac{k_z}{w-1}, b\right) \middle| k_z \in (-w+1, w-1] \subset \mathbb{Z} \wedge b \in \mathbb{Z} \wedge p \neq \varnothing \right\}, \\ S^{KV}_{w,h} &\stackrel{\text{def}}{=} \left\{ p = p^{V}_{w,h}\left(\frac{k_z}{h-1}, b\right) \middle| k_z \in [-h+1, h-1) \subset \mathbb{Z} \wedge b \in \mathbb{Z} \wedge p \neq \varnothing \right\}. \end{aligned} \tag{23}$$

The procedure for calculating sums over patterns from the set $S^{K}_{w,h}$ in the image $I_{w,h}$ using the $ASD2$ algorithm completely repeats the scheme of applying the Brady–Yong algorithm [1].

The image $(I_{w,h})$ is padded with zeros up to a size of $I_{w,h+w}$. The $ASD2$ algorithm is applied to this image. The same pair of operations is performed with the image reflected on the second axis, the transposed image, and the transposed and reflected image. The resulting four sets of sums cover the set $S^{K}_{w,h}$. The total complexity of the algorithm in this case is $O(w^{8/3} + h^{8/3})$.

6. Discussion

The *Build_Gkchp* function in the $ASD2$ algorithm can be replaced without affecting its functionality. Thus, the list of patterns to be summed can be varied. This is interesting both in terms of further introduction of sparsity into the pattern set in cases when high-resolution results are not required and in terms of computing patterns with $k \leq 0$ and with $k \geq 0$ in a single pass. The computational complexity of such modifications to the $ASD2$ algorithm is not discussed in this work. However, it is clear that for majorization of summations, the $AS4R$ algorithm relies not on the internal structure of the pattern set but only on its cardinality.

It is also worth noting that the $ASD2$ algorithm itself does not require the condition $w = 8^q$ to be satisfied, as this condition is used only to construct a complexity estimate. As for the original Brady–Yong algorithm, its modification supporting arbitrary array sizes was published only in 2021 [66]. A solution to this problem had already been announced nine years earlier in [67], but the proposed algorithm was based on factoring of the array size and was not efficient for sizes with large prime factors. As for the $ASD2$ algorithm, it would be interesting to evaluate its complexity for arbitrary image sizes.

It is also interesting to consider the generalization of the $ASD2$ algorithm to three-dimensional images. Generalizations of the Brady–Yong algorithm for summations over lines and planes in three-dimensional images were first published by T.-K. Wu and M. Brady in 1998 [68], but this publication in the proceedings of a symposium on image processing and character recognition went unnoticed for a long time. Both algorithms have been repeatedly reinvented [26,69].

It is worth noting that the estimation of the number of DSLS in this work is not the first of its kind. A more accurate estimation was obtained in [70], but the reasoning presented in Section 3.2 appears to be simpler to us while still illustrating one of the key ideas of the paper.

Another important area for future research is the numerical comparison of the various algorithms investigated in this study, including the Brady–Yong and Khanipov algorithms. To conduct such a comparison, it is necessary to reconstruct the Khanipov algorithm based on the provided scheme. Ensuring a fair assessment under identical conditions requires

the careful implementation of all algorithms. It is particularly important to pay attention to algorithms based on the pseudo-polar Fourier transform, as their performance can be significantly influenced by the choice of libraries. Additionally, it would be interesting to compare the accuracy of these algorithm classes using a fixed data discretization model.

Finally, in certain applications, such as tomographic reconstruction, it is necessary to obtain not the projection operator (A) calculated by the aforementioned algorithms but its "inverse". The use of quotation marks in this context is to highlight that in reconstruction problems, a regularized problem is typically stated:

$$\|Ax - y\|^2 + R(x) \to \min_x, \tag{24}$$

where x is the target image, y is the observed image, and R is a regularizer. One approach to tackle this optimization problem involves gradient-descent-type iterative algorithms [71]. These algorithms require repeated computations of the adjoint operator (A^T). From this perspective, fast algorithms for computing the adjoint operator of the Hough transform are more significant than fast algorithms for direct inverse transform. It is feasible to modify the considered algorithms to efficiently compute the image $A^T z$, and this aspect warrants further research.

Author Contributions: Conceptualization, D.N. and A.K.; software, E.L.; validation, E.L., A.M. and I.F.; writing—original draft preparation, D.N. and E.E.; writing—review and editing, D.N., A.K., E.L. and A.M.; supervision, I.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DSLS	Digital straight line segment
FGHT	Fast generalized Hough transform
FHT	Fast Hough transform
HT	Hough transform

References

1. Brady, M.L.; Yong, W. Fast Parallel Discrete Approximation Algorithms for the Radon Transform. In Proceedings of the SPAA'92: Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, USA, 29 June–1 July 1992 ; pp. 91–99. [CrossRef]
2. Duda, R.O.; Hart, P.E. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Commun. ACM* **1972**, *15*, 11–15. [CrossRef]
3. Nikolaev, D.P.; Nikolayev, P.P. Linear color segmentation and its implementation. *Comput. Vis. Image Underst.* **2004**, *94*, 115–139. [CrossRef]
4. Shi, L.; Funt, B. Dichromatic illumination estimation via Hough transforms in 3D. In Proceedings of the European Conference on Colour in Graphics, Imaging, and Vision (IS&T, 2008), Barcelona, Spain, June 2008; pp. 259–262.
5. Berman, D.; Treibitz, T.; Avidan, S. Air-light estimation using haze-lines. In Proceedings of the IEEE International Conference on Computational Photography, Stanford, CA, USA, 12–14 May 2017; pp. 1–9. [CrossRef]
6. Kunina, I.A.; Gladilin, S.A.; Nikolaev, D.P. Blind compensation of radial distortion in a single image using fast Hough transform. *Comput. Opt.* **2016**, *40*, 395–403. [CrossRef]
7. Yang, S.; Rong, J.; Huang, S.; Shang, Z.; Shi, Y.; Ying, X.; Zha, H. Simultaneously vanishing point detection and radial lens distortion correction from single wide-angle images. In Proceedings of the 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO), Qingdao, China, 3–7 December 2016; pp. 363–368. [CrossRef]
8. Chang, Y.; Bailey, D.; Le Moan, S. Lens distortion correction by analysing peak shape in Hough transform space. In Proceedings of the 2017 International Conference on Image and Vision Computing New Zealand (IVCNZ), Christchurch, New Zealand, 4–6 December 2017; pp. 1–6. [CrossRef]

9. Aminuddin, N.S.; Ibrahim, M.M.; Ali, N.M.; Radzi, S.A.; Saad, W.H.M.; Darsono, A.M. A new approach to highway lane detection by using Hough transform technique. *J. Inf. Commun. Technol.* **2017**, *16*, 244–260. [CrossRef]
10. Panfilova, E.I.; Shipitko, O.S.; Kunina, I.A. Fast Hough Transform-Based Road Markings Detection For Autonomous Vehicle. In Proceedings of the SPIE 11605, Thirteenth International Conference on Machine Vision (ICMV 2020), Rome, Italy, 2–6 November 2020; Volume 11605, pp. 671–680. [CrossRef]
11. Jahan, R.; Suman, P.; Singh, D.K. Lane detection using canny edge detection and hough transform on raspberry Pi. *Int. J. Adv. Res. Comput. Sci.* **2018**, *9*, 85–89. [CrossRef]
12. Guan, J.; An, F.; Zhang, X.; Chen, L.; Mattausch, H.J. Energy-Efficient Hardware Implementation of Road-Lane Detection Based on Hough Transform with Parallelized Voting Procedure and Local Maximum Algorithm. *IEICE Trans. Inf. Syst.* **2019**, *E102.D*, 1171–1182. [CrossRef]
13. Thongpan, N.; Rattanasiriwongwut, M.; Ketcham, M. Lane Detection Using Embedded System. *Int. J. Comput. Internet Manag.* **2020**, *28*, 46–51.
14. Schwartz, R.; Dodge, J.; Smith, N.A.; Etzioni, O. Green AI. *Commun. ACM* **2020**, *63*, 54–63. [CrossRef]
15. Hartl, A.; Reitmayr, G. Rectangular target extraction for mobile augmented reality applications. In Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), Tsukuba, Japan, 11–15 November 2012; pp. 81–84.
16. Puybareau, É.; Géraud, T. Real-Time Document Detection in Smartphone Videos. In Proceedings of the 2018 25th IEEE International Conference on Image Processing (ICIP), Athens, Greece, 7–10 October 2018; pp. 1498–1502. [CrossRef]
17. Tropin, D.V.; Ilyuhin, S.A.; Nikolaev, D.P.; Arlazarov, V.V. Approach for Document Detection by Contours and Contrasts. In Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, 10–15 January 2021; pp. 9689–9695.[CrossRef]
18. Gatos, B.; Perantonis, S.J.; Papamarkos, N. Accelerated Hough transform using rectangular image decomposition. *Electron. Lett.* **1996**, *32*, 730–732. [CrossRef]
19. Singh, C.; Bhatia, N.; Kaur, A. Hough transform based fast skew detection and accurate skew correction methods. *Pattern Recognit.* **2008**, *41*, 3528–3546. [CrossRef]
20. Bezmaternykh, P.V.; Nikolaev, D.P. A document skew detection method using fast Hough transform. In Proceedings of the SPIE 11433, Twelfth International Conference on Machine Vision (ICMV 2019), Amsterdam, The Netherlands, 16–18 November 2019; Volume 11433, pp. 132–137. [CrossRef]
21. Gao, Y.P.; Li, Y.M.; Hu, Z.Y. Skewed text correction based on the improved Hough transform. In Proceedings of the 2011 International Conference on Image Analysis and Signal Processing, Wuhan, China, 21–23 October 2011; pp. 368–372. [CrossRef]
22. Limonova, E.; Bezmaternykh, P.; Nikolaev, D.; Arlazarov, V. Slant rectification in Russian passport OCR system using fast Hough transform. In Proceedings of the SPIE 10341, Ninth International Conference on Machine Vision (ICMV 2016), Nice, France, 18–20 November 2016 ; Volume 10341, pp. 127–131. [CrossRef]
23. Martynov, S.I.; Bezmaternykh, P.V. Aztec core symbol detection method based on connected components extraction and contour signature analysis. In Proceedings of the SPIE 11433, Twelfth International Conference on Machine Vision (ICMV 2019), Amsterdam, The Netherlands, 16–18 November 2019; Volume 11433, pp. 27–34. [CrossRef]
24. Bulatov, K.B.; Chukalina, M.V.; Nikolaev, D.P. Fast X-ray sum calculation algorithm for computed tomography. *Bull. South Ural. State Univ. Ser. Math. Model. Program. Comput. Softw.* **2020**, *13*, 95–106. [CrossRef]
25. Dolmatova, A.V.; Chukalina, M.V.; Nikolaev, D.P. Accelerated FBP for Computed Tomography Image Reconstruction. In Proceedings of the 2020 IEEE International Conference on Image Processing, Abu Dhabi, United Arab Emirates, 25–28 October 2020; pp. 3030–3034. [CrossRef]
26. Marichal-Hernandez, J.G.; Gómez-Cárdenes, Ó.; González, F.L.R.; Kim, D.H.; Rodríguez-Ramos, J.M. Three-dimensional multiscale discrete Radon and John transforms. *Opt. Eng.* **2020**, *59*, 093104. [CrossRef]
27. Bulatov, K.; Chukalina, M.; Buzmakov, A.; Nikolaev, D.; Arlazarov, V.V. Monitored Reconstruction: Computed Tomography as an Anytime Algorithm. *IEEE Access* **2020**, *8*, 110759–110774. [CrossRef]
28. Sheshkus, A.; Chirvonaya, A.; Nikolaev, D.; Arlazarov, V.L. Vanishing point detection with direct and transposed fast Hough transform inside the neural network. *Comput. Opt.* **2020**, *44*, 737–745. [CrossRef]
29. Lin, Y.; Pintea, S.L.; van Gemert, J.C. Deep Hough-Transform Line Priors. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, 23–28 August 2020*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12367, pp. 323–340. [CrossRef]
30. Han, Q.; Zhao, K.; Xu, J.; Cheng, M.M. Deep Hough Transform for Semantic Line Detection. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, 23–28 August 2020*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12354, pp. 249–265. [CrossRef]
31. Teplyakov, L.; Kaymakov, K.; Shvets, E.; Nikolaev, D. Line detection via a lightweight CNN with a Hough Layer. In Proceedings of the SPIE 11605, Thirteenth International Conference on Machine Vision (ICMV 2020), Rome, Italy, 2–6 November 2020; Volume 11605, pp. 376–385. [CrossRef]
32. Zhao, K.; Han, Q.; Zhang, C.B.; Xu, J.; Cheng, M.M. Deep Hough Transform for Semantic Line Detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 4793–4806. [CrossRef]
33. Zhao, H.; Zhang, Z. Improving Neural Network Detection Accuracy of Electric Power Bushings in Infrared Images by Hough Transform. *Sensors* **2020**, *20*, 2931. [CrossRef]

34. Nabil, A. Combination of Hough Transform and Neural Network on recognizing mathematical symbols. In Proceedings of the 2021 8th International Conference on ICT & Accessibility (ICTA), Tunis, Tunisia, 8–10 December 2021; pp. 1–6. [CrossRef]
35. Jin, K.H.; McCann, M.T.; Froustey, E.; Unser, M. Deep Convolutional Neural Network for Inverse Problems in Imaging. *IEEE Trans. Image Process.* **2017**, *26*, 4509–4522. [CrossRef]
36. Adler, J.; Öktem, O. Learned Primal-Dual Reconstruction. *IEEE Trans. Med. Imaging* **2018**, *37*, 1322–1332. [CrossRef]
37. Jiao, F.; Gui, Z.; Li, K.; Hong, S.; Wang, Y.; Liu, Y.; Zhang, P. A Dual-Domain CNN-Based Network for CT Reconstruction. *IEEE Access* **2021**, *9*, 71091–71103. [CrossRef]
38. Hough, P.V.C. Machine Analysis of Bubble Chamber Pictures. In *International Conference on High Energy Accelerators and Instrumentation*; CERN: Geneva, Switzerland , 1959; Volume 590914, pp. 554–558.
39. Hough, P.V.C. Method and Means for Recognizing Complex Patterns. US 3069654 A, 18 December 1962.
40. Hart, P.E. How the Hough transform was invented [DSP History]. *IEEE Signal Process. Mag.* **2009**, *26*, 18–22. [CrossRef]
41. Rosenfeld, A. *Picture Processing by Computer*; Academic Press: New York, NY, USA, 1969.
42. Deans, S.R. Hough Transform from the Radon Transform. *IEEE Trans. Pattern Anal. Mach. Intell.* **1981**, *PAMI-3*, 185–188. [CrossRef] [PubMed]
43. Radon, J. Über die Bestimmung von Funktionen durch ihre Integralwerte längs gewisser Mannigfaltigkeiten. *Berichte Über Verhandlungen Sächs. Akad. Wiss.* **1917**, *69*, 262–277.
44. Illingworth, J.; Kittler, J. A survey of the hough transform. *Comput. Vision Graph. Image Process.* **1988**, *44*, 87–116. [CrossRef]
45. Brady, M.L. A fast discrete approximation algorithm for the Radon transform. *Siam J. Comput.* **1998**, *27*, 91–99. [CrossRef]
46. Götz, W.A.; Druckmüller, H.J. A fast digital Radon transform—An efficient means for evaluating the Hough transform. *Pattern Recognit.* **1995**, *28*, 1985–1992. [CrossRef]
47. Götz, W.A.; Druckmüller, H.J. A fast digital Radon transform—An efficient means for evaluating the Hough transform. *Pattern Recognit.* **1996**, *29*, 711–718. [CrossRef]
48. Vuillemin, J.E. Fast linear Hough transform. In Proceedings of the IEEE International Conference on Application Specific Array Processors (ASSAP'94), San Francisco, CA, USA, 22–24 August 1994; pp. 1–9. [CrossRef]
49. Karpenko, S.M.; Ershov, E.I. Analysis of Properties of Dyadic Patterns for the Fast Hough Transform. *Probl. Inf. Transm.* **2021**, *57*, 292–300. [CrossRef]
50. Averbuch, A.; Coifman, R.R.; Donoho, D.L.; Israeli, M.; Shkolnisky, Y.; Sedelnikov, I. A Framework for Discrete Integral Transformations II—The 2D Discrete Radon Transform. *Siam J. Sci. Comput.* **2008**, *30*, 785–803. [CrossRef]
51. Nikolaev, D.P.; Karpenko, S.M.; Nikolayev, I.P. Hough Transform: Underestimated Tool In The Computer Vision Field. In Proceedings of the 22nd European Conference on Modelling and Simulation, Nicosia, Cyprus, 3–6 June 2008; pp. 238–243. [CrossRef]
52. Levi, O.; Efros, B.A. A new fast algorithm for exact calculation of the discrete 2-d and 3-d x-ray transform. In *Advances in Computational Methods in Sciences and Engineering 2005: Selected Papers from the International Conference of Computational Methods in Sciences and Engineering 2005 (ICCMSE 2005)*; *Lecture Series on Computer and Computational Sciences*; Brill Academic Publishers: Leiden, The Netherlands, 2005; Volume 4, pp. 319–322.
53. Rosenfeld, A. Digital Straight Line Segments. *IEEE Trans. Comput.* **1974**, *C-23*, 1264–1269. [CrossRef]
54. Khanipov, T.M. Computational complexity lower bounds of certain discrete Radon transform approximations. *arXiv* **2018**, arXiv:1801.01054. [CrossRef]
55. Khanipov, T.M. Ensemble computation approach to the Hough transform. *arXiv* **2018**, arXiv:1801.01054. [CrossRef]
56. Yang, D. Fast discrete Radon transform and 2-D discrete Fourier transform. *Electron. Lett.* **1990**, *26*, 550–551. [CrossRef]
57. Gertner, I. A new efficient algorithm to compute the two-dimensional discrete Fourier transform. *IEEE Trans. Acoust. Speech Signal Process.* **1988**, *36*, 1036–1050. [CrossRef]
58. Ershov, E.I.; Khanipov, T.M.; Shvets, E.A.; Nikolaev, D.P. Generation algorithms of fast generalized hough transform. In Proceedings of the 31st European Conference on Modelling and Simulation, ECMS 2017, Budapest, Hungary, 23–26 May 2017; pp. 534–538.
59. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; W. H. Freeman and Co.: New York, NY, USA, 1979; p. 338.
60. Ikenaga, Y.; Yamaguchi, K. Proposal of Greedy Random for Ensemble Computation. *Jsai Tech. Rep. SIG-FPAI* **2022**, *120*, 7–10. [CrossRef]
61. Steiner, J. Einige Gesetze über die Theilung der Ebene und des Raumes. *J. Füur Reine Angew. Math.* **1826**, *1*, 349–364.
62. Arlazarov, V.L.; Dinitz, Y.A.; Kronrod, M.A.; Faradzhev, I.A. On economical construction of the transitive closure of an oriented graph. *Dokl. Akad. Nauk SSSR* **1970**, *194*, 487–488. (In Russian)
63. Soshin, K.V.; Nikolaev, D.P.; Gladilin, S.A.; Ershov, E.I. Acceleration of summation over segments using the fast Hough transformation pyramid. *Bull. South Ural. State Univ. Ser. Math. Model. Program. Comput. Softw.* **2020**, *13*, 129–140. [CrossRef]
64. Lindenbaum, M.; Koplowitz, J. A new parameterization of digital straight lines. *IEEE Trans. Pattern Anal. Mach. Intell.* **1991**, *13*, 847–852. [CrossRef]
65. Shepp, L.A.; Logan, B.F. The Fourier reconstruction of a head section. *IEEE Trans. Nucl. Sci.* **1974**, *21*, 21–43. [CrossRef]
66. Anikeev, F.A.; Raiko, G.O.; Limonova, E.E.; Aliev, M.A.; Nikolaev, D.P. Efficient Implementation of Fast Hough Transform Using CPCA Coprocessor. *Program. Comput. Softw.* **2021**, *47*, 335–343. [CrossRef]

67. Marichal-Hernandez, J.G.; Lüke, J.P.; González, F.L.R.; Rodríguez-Ramos, J.M. Fast approximate 4-D/3-D discrete radon transform for lightfield refocusing. *J. Electron. Imaging* **2012**, *21*, 023026. [CrossRef]
68. Wu, T.K.; Brady, M.L. A fast approximation algorithm for 3D image reconstruction. In Proceedings of the 1998 International Computer Symposium. Workshop in Image Processing and Character Recognition, Tainan, Taiwan, 17–19 December 1998; pp. 213–220.
69. Ershov, E.I.; Terekhin, A.P.; Nikolaev, D.P. Generalization of the Fast Hough Transform for Three-Dimensional Images. *J. Commun. Technol. Electron.* **2018**, *63*, 626–636. [CrossRef]
70. Koplowitz, J.; Lindenbaum, M.; Bruckstein, A. The number of digital straight lines on an N*N grid. *IEEE Trans. Inf. Theory* **1990**, *36*, 192–197. [CrossRef]
71. Tian, Z.; Jia, X.; Yuan, K.; Pan, T.; Jiang, S.B. Low-dose CT reconstruction via edge-preserving total variation regularization. *Phys. Med. Biol.* **2011**, *56*, 5949–5967. [CrossRef] [PubMed]