Pattern Analysis and Applications (2024) 27:113
https://doi.org/10.1007/s10044-024-01332-6

ORIGINAL ARTICLE

Methods for calculating gliding-box lacunarity efficiently on large datasets

Bálint Barna H. Kovács[1] · **Miklós Erdélyi**[1]

Received: 16 May 2024 / Accepted: 29 August 2024

Abstract
Lacunarity has proven to be a useful, multifaceted tool for image analysis in several different scientific fields, from geography to virology, which has lent increasing importance to the lacunarity analysis of large datasets. It can be most reliably calculated with the so-called gliding-box method, but the evaluation process can be exceedingly time-consuming and unviable as this algorithm is not designed to operate on large datasets. Here we introduce two novel methods that can calculate gliding-box lacunarity orders of magnitude faster than the original method without any loss of accuracy. We compare these methods with the original as well as with two already existing optimized methods based on runtime memory usage and complexity. The application of all five methods for both 2D and 3D datasets analysis confirms that each of the four optimized methods are orders of magnitude faster than the original one, but each has its advantages and limitations.

Keywords Multi-scale lacunarity · Fractal analysis · Pattern recognition

1 Introduction

In the interdisciplinary field of data science, the numerical quantification of the texture and spatial heterogenity of datasets has yielded invaluable results. Among fractal based mathematical approaches the calculation of lacunarity has proven itself in several scientific fields from fractal analysis [1, 2] and geography [3, 4] through astronomy [5] and micro-CT analysis [6] to oncology [7–11], neuroscience [12], single molecule localization microscopy [13] and COVID-19 research [14]. Lacunarity, originating from Mandelbrot's seminal book "The fractal geometry of nature" [15], was introduced as a quantitative measure of the degree of variability in the size and distribution of empty spaces or voids in a dataset. As it can be calculated for any specific void size, it is a multi-scaled measure of the homogeneity of datasets, and can be used in the numerous different fields listed above. One of the most well-known, consistent and accurate ways to calculate the lacunarity of a dataset is the gliding-box method developed by Alain and Cloitre to characterize binary images [16]. This method measures lacunarity by analyzing the variability of occupied spaces in all possible ε sized boxes for any desired possible ε value. A significant drawback of the original algorithm is its computational intensity. Although computational power is increasing exponentially, the analysis of large datasets can pose a challenge with the original gliding-box method. For this reason, several different approaches have been introduced to optimize the original algorithm [17–19]; however, they can be challenging to find and are not in wide-spread use.

In this paper, we have selected two existing, highly efficient, algorithms and introduced two new methods that can calculate the gliding-box lacunarity value several orders of magnitude faster than the original algorithm. Our goal is to make these optimized algorithms more accessible to the larger scientific community while comparing them to the original method and to each other in terms of runtime, memory usage and complexity.

✉ Bálint Barna H. Kovács
h.kovacs.balint.barna@szte.hu

Miklós Erdélyi
erdelyi.miklos@szte.hu

[1] Department of Optics and Quantum Electronics, University of Szeged, Dóm square 9, Szeged 6720, Hungary

Published online: 13 September 2024

2 Materials and methods

The calculation of lacunarity, using any method based on the original gliding-box algorithm, can be divided into three steps. First, we need to condition the dataset so that the specific algorithm can run efficiently. This step needs to be performed only once per dataset, thus it requires an insignificant number of operations compared to the second step, when we calculate the masses of the boxes, i.e. count the object pixels or data points in all possible "ε" sized boxes. This step needs to be done once for every desired "ε" box size, making it the most computationally intensive part of the original gliding-box algorithm. The third step is the actual calculation of the lacunarity value from the box masses. This needs to be done for each "ε" box size; however, this third step is not computationally intensive and can be separated from the first two steps. Here we will describe in detail five different algorithms performing the first and second steps of the lacunarity calculation process while comparing their properties. We will also show an optimized version of the third step of the lacunarity calculation. For the sake of simplicity, we will compare the different algorithms by evaluating grayscale images of the same size in all dimensions. We define an object pixel on a grayscale image as any pixel that has a value larger than 0. To implement a fair comparison between the algorithms we will measure single thread performance and also investigate the scaling factor for multithreading. It is important to mention that all studied algorithms can be parallelized allowing greater efficiency. To discuss the applied algorithms consistently we will introduce the following notations (See Table 1):

The third step of the calculation, responsible for obtaining the lacunarity values from the box masses, was originally performed by calculating the mass distribution for each box size. In the case of binary images, this is a doable but memory intensive process. In the case of grayscale images, it can be hard to assess the highest possible box mass. Tolle introduced an optimized version [17], which directly calculates the lacunarity value at a given box size from the box masses for 2D datasets (Eq. 1).

$$\Lambda\ (\epsilon\ ) = \frac{B(N,\epsilon\ ) \cdot \sum_{i,j=1}^{N-\epsilon+1} (M\ (i,j))^2}{\left(\sum_{i,j=1}^{N-\epsilon+1} M\ (i,j)\right)^2} \tag{1}$$

Here $\mathrm{M\,(i,j)}$ represents the mass of the box in the i-th row and j-th column, $\mathrm{B(N,\ \ )}$ is the number of boxes with a size ε on an N by N image, which is $B(N,\varepsilon) = (N-\varepsilon+1)^2$. For 3D datasets the equation can be written as:

$$\Lambda\ (\epsilon\ ) = \frac{B(N,\epsilon\ ) \cdot \sum_{i,j,k=1}^{N-\epsilon+1} (M\ (i,j,k))^2}{\left(\sum_{i,j,k=1}^{N-\epsilon+1} M\ (i,j,k)\right)^2}\ , \tag{2}$$

where $\mathrm{M\,(i,j,k)}$ represents the mass of the box in the i-th row and j-th column and k-th in the third dimension, which we will call page; $\mathrm{B(N,\ \ )}$ is the number of boxes with a size ε on an N by N by N image, which is $\mathrm{B\,(N,\ \ ) = (N - \ \ + 1)^3}$.

Table 1 Table of notations

N	The size of the grayscale image in all dimensions.	A positive integer.
ε	The size of the box.	A positive integer less or equal to the size of the grayscale image.
$B(\varepsilon)$	The number of ε size boxes.	A positive integer.
$\Lambda(\varepsilon)$	Lacunarity value at a box size ε.	A double-precision floating point number.
D	The dimensions of the grayscale image.	A positive integer. In this paper 2 or 3.
S	The original grayscale image.	A D dimensional N length array of integers.
$\widehat{S}$	The Fast Fourier Transform of the original grayscale image.	A D dimensional N length array of double-precision complex floating-point numbers.
E	One-dimensional ε sized kernel.	A one-dimensional N length array of double-precision floating-point numbers.
$\widehat{E}$	The Fast Fourier Transform of the one-dimensional ε sized kernel.	A one-dimensional N length array of double-precision complex floating-point numbers.
M	The box masses.	A D dimensional N-ε + 1 length array of integers or double-precision floating-point numbers.
V	Value of each object pixel.	A one-dimensional array of integers with the length of the number of total object pixels.
P_r	Row position of each object pixel.	A one-dimensional array of integers with the length of the number of total object pixels.
P_c	Column position of each object pixel.	A one-dimensional array of integers with the length of the number of total object pixels.
P_p	Page position of each object pixel.	A one-dimensional array of integers with the length of the number of total object pixels.
I	Extended integral image of the original binary image.	A D dimensional $N+1$ length array of integers or double-precision floating-point numbers.

2.1 Original gliding-box method

The original gliding box method was introduced by Alain and Cloitre [16]. Data conditioning consists of converting the grayscale image into a datatype capable of handling the addition of a large number of elements without overflow. To calculate the box masses, all elements in each box are added.

2.1.1 2D algorithm

For 2D images, the box mass calculation process is shown in Fig. 1. There are $(N-\epsilon+1)^2$ boxes and ϵ^2 elements in each box. This means the operation cost of the calculation is:

$$O_{Original\,2D}(N,\epsilon) \sim (N-\epsilon+1)^2 \cdot \epsilon^2 \tag{3}$$

2.1.2 3D algorithm

For 3D images, there are $(N-\epsilon+1)^3$ boxes and there are ϵ^3 elements in each box. This means the operation cost of the calculation is:

$$O_{Original\,3D}(N,\epsilon) \sim (N-\epsilon+1)^3 \cdot \epsilon^3 \tag{4}$$

2.1.3 Space complexity

The space complexity of an algorithm describes its theoretical memory usage in relation to input size. For the sake of consistency, we examine space complexity in relation to image size N. It can be calculated by examining the memory required by the input and auxiliary spaces of the code. The input of the original algorithm is the D dimensional image. The auxiliary space consists of the outputs which are the box masses.

Input space: N^D

Auxiliary space: $(N-\epsilon+1)^D$

Total: $N^D+(N-\epsilon+1)^D$

Expressed using big O notation the space complexity is: $O\left(N^D\right)$

2.2 Sides gliding-box method

The algorithm that we refer to as the sides gliding-box method was introduced by Backes [18]. The box mass calculation is optimized by utilizing the overlapping of consecutive boxes through the gliding process. Data conditioning consists of converting the grayscale image into a datatype capable of handling the addition of a large number of elements without overflow.

2.2.1 2D algorithm

In the 2D version of the algorithm, to calculate the mass of a box adjacent to an already calculated box, only the edges that do not overlap need to be visited, as it can be seen in Fig. 2. This means that on average 2ϵ elements need to be added up in each box instead of ϵ^2, hence the operation cost will total:

$$O_{Side\,2D}(N,\epsilon) \sim (N-\epsilon+1)^2 \cdot 2 \cdot \epsilon \tag{5}$$

2.2.2 3D algorithm

In the 3D version of the algorithm, to calculate the mass of a box that is adjacent to an already calculated box only the non-overlapping faces need to be visited. This means that on average only $2\epsilon^2$ elements need to be added up in each box instead of ϵ^3, hence the operating cost will total:

$$O_{Side\,3D}(N,\epsilon) \sim (N-\epsilon+1)^3 \cdot 2 \cdot \epsilon^2 \tag{6}$$

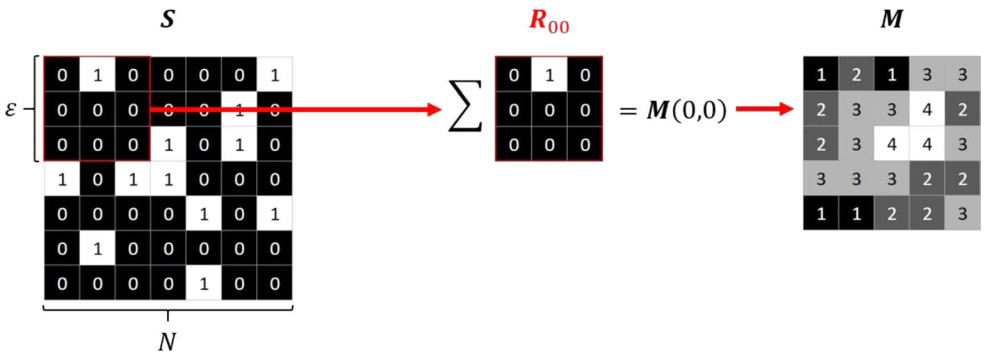

Fig. 1 Flowchart of the original gliding-box method. Original gliding-box method on an image of size $N=7$ at a box size of $\varepsilon=3$. From the original image (S) all possible 3 by 3 regions of interest (R_{ij}) are taken and their value sums result in the box masses (M)

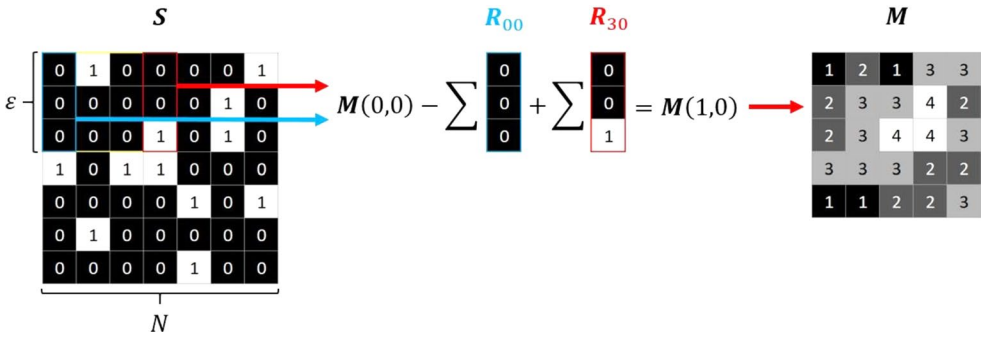

Fig. 2 Flowchart of the sides gliding-box method. Sides gliding-box method on an image of $N=7$ at a box size of $\varepsilon=3$. From the original image (S) the box sizes (M) are iteratively calculated by adding and subtracting 3 by 1 regions of interest (R_{ij}) that do not overlap on consecutive boxes

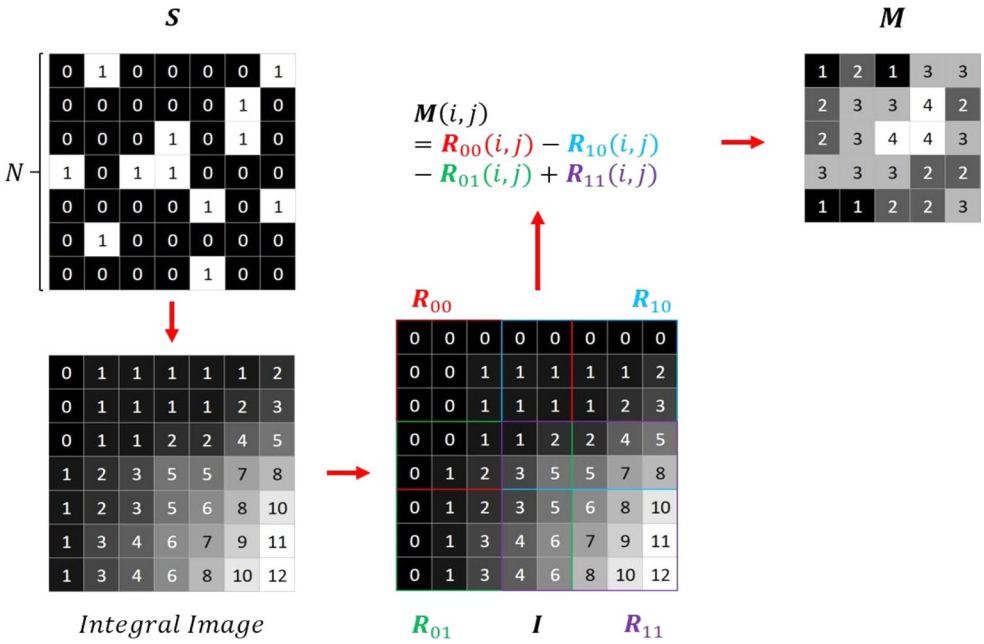

Fig. 3 Flowchart of the integral gliding-box method. Integral gliding-box method on an image of $N=7$ at a box size of $\varepsilon=3$. From the original image (S) an integral image is generated representing the sum value of all elements to the left and above from a given pixel (including that pixel). This integral image is extended by a row and column of zeros (I). The box masses (M) are calculated by adding the four M sized subarrays from the four corners of the extended integral image with appropriate signs

2.2.3 Space complexity

The input of the sides algorithm is the original D dimensional image. The auxiliary space consists only of the outputs, which are the box masses.

Input space: N^D
Auxiliary space: $(N-\epsilon+1)^D$
Total: $N^D+(N-\epsilon+1)^D$

Expressed using big O notation the space complexity is: $O\left(N^D\right)$

2.3 Integral method

The algorithm that we refer to as the integral gliding-box method was first shown for lacunarity calculation by Williams [19]. This method optimizes the original algorithm utilizing special data structuring, called an integral image, allowing for the fast computation of rectangular structures [20]. Data conditioning consists of converting the grayscale image into a data type capable of handling large numbers and the creation of the integral image.

2.3.1 2D algorithm

For the 2D version of the algorithm, the generation of an integral image from the original grayscale image is done in the following manner: At each (i, j) location in the integral image, the value equals the sum value of all pixels with indexes less than or equal to (i, j). To make the box mass calculation a bit more straightforward, the integral image can be extended by shifting all values to a higher row and column position by one, i.e. by introducing a new first row and column consisting of zeros only. On this new extended integral image all the $(N-\epsilon+1)^2$ box masses can be calculated by taking four $N-\epsilon+1$ by $N-\epsilon+1$ arrays from the four corners of the extended integral image, adding the two arrays originating from the corners on the main diagonal and subtracting the arrays originating from the corners on the secondary diagonal shown in Fig. 3. The operation cost in this case equals:

$$O_{Integral\,2D}\,(N,\epsilon\,)\;\sim\;(N-\epsilon\,+1)^2\cdot\,4 \tag{7}$$

2.3.2 3D algorithm

For the 3D version of the integral method, the generation of an integral image from the original grayscale image is done in the following manner: At each (i, j,k) location in the integral image, the value corresponds to the sum value of all pixels with indexes less than or equal to (i, j,k). To make the box mass calculation a bit more straightforward, the integral image can be extended by shifting all values to a higher row, column and page position by one, i.e. by introducing a new first row, column and page consisting of zeros only. On this new extended integral image all the $(N-\epsilon\,+1)^3$ box masses can be calculated by taking eight $N-\epsilon\,+1$ by $N-\epsilon\,+1$ by $N-\epsilon\,+1$ arrays from the eight corners of the extended integral image and adding them with appropriate signs. The sign of each array can be visualized using the following identity:

$$(x-1)^3=\;x^3-\;3x^2+\;3x-1\;\;, \tag{8}$$

where 1 represents the corner containing the (0, 0, 0) element having a negative sign. The $3x$ represents the three corners containing the (N+1, 0, 0), the (0, N+1, 0) and the (0, 0, N+1) elements. These corners neighbor the corner containing the (0,0,0) element and have positive signs. The $3x^2$ stands for the three corners containing the (0, N+1, N+1), the (N+1, 0, N+1) and the (N+1, N+1, 0) elements. These corners have a negative sign. Finally, the x^3 corresponds to the corner containing the (N+1, N+1, N+1) element having a positive sign. This analogy can be used for higher dimensions too. The operation cost involved is:

$$O_{Integral\,3D}\,(N,\epsilon\,)\;\sim\;(N-\epsilon\,+1)^3\cdot\,8 \tag{9}$$

2.3.3 Space complexity

The input of the integral algorithm is the D dimensional extended integral image created from the original image. The auxiliary space only contains the outputs which are the box masses.

Input space: $(N+1)^D$
Auxiliary space: $(N-\epsilon\,+1)^D$
Total: $(N+1)^D+(N-\epsilon\,+1)^D$

Expressed using big O notation the space complexity is: $O\left(N^D\right)$

2.4 Trace gliding-box method

We named the first algorithm to be introduced here as trace gliding-box method. This algorithm optimizes the original method by working backwards: instead of counting elements in boxes, it increases the value of boxes that contain a given object pixel by its value. Data conditioning means the creation of an empty box mass array and the collection of the object pixel indices and values.

2.4.1 2D algorithm

For the 2D trace method, the values as well as the row and column indices of all object pixels V, P_r and P_c are collected. The box masses are calculated by generating an array having the size of the box masses ($N-\epsilon\,+1$ by $N-\epsilon\,+1$) with all elements being zeros, and increasing the value of the boxes containing a given object pixel by the value of that object pixel in the original grayscale image shown in Fig. 4. After this is done for each individual object pixel, we get the box masses. When the number of object pixels is C ,the operation cost equals:

$$O_{Trace\,2D}\,(N,\epsilon\,)\;\sim\;C\cdot\,\epsilon^{\,2} \tag{10}$$

2.4.2 3D algorithm

For the 3D trace gliding-box algorithm, the values as well as the row, column and page indices of all object pixels V, P_r, P_c and P_p are collected. The box masses are calculated by generating an array having the size of the box masses ($N-\epsilon\,+1$ by $N-\epsilon\,+1$ by $N-\epsilon\,+1$) with all elements being zeros and increasing the value of the boxes containing a given object pixel by the value of the pixel in the original grayscale image. After this is done for each individual object pixel, we get the box masses. When the number of object pixels is C, this results in the following operation cost:

$$O_{Trace\,3D}\,(N,\epsilon\,)\;\sim\;C\cdot\,\epsilon^{\,3} \tag{11}$$

2.4.3 Space complexity

The inputs of the trace algorithm are the $D+1$ vectors containing the indices and values of the object pixels. The auxiliary space contains the box masses only.

Input space: $(D+1)\cdot\,C$
Auxiliary space: $(N-\epsilon\,+1)^D$

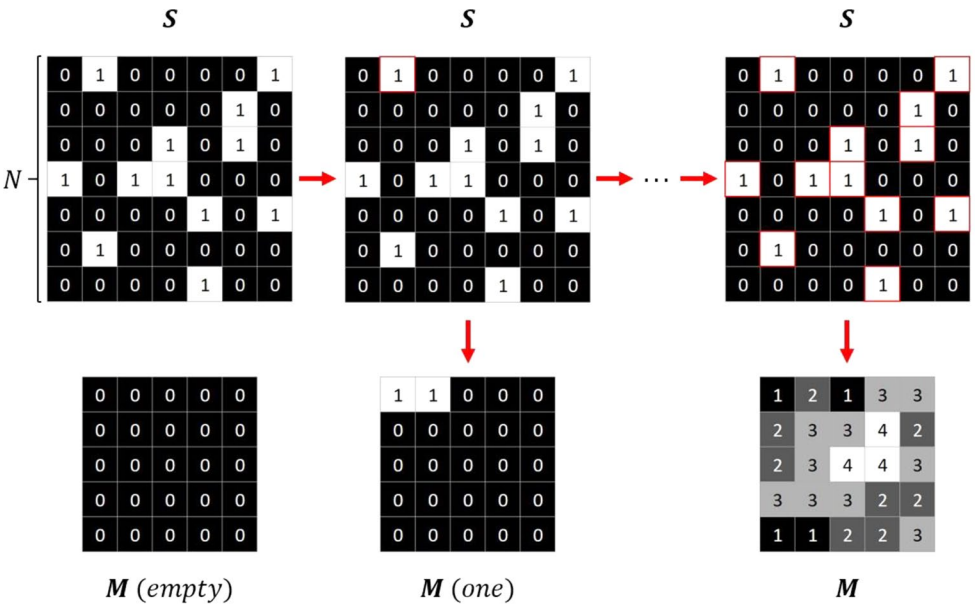

Fig. 4 Flowchart of the trace gliding-box method. Trace gliding-box method on an image of $N=7$ at a box size of $\varepsilon=3$. In an empty array having the size of the box masses, all elements influenced by an object pixel from the original image (S) are increased by this pixel's value for every object pixel resulting in the box masses (M)

Total: $(D+1)\cdot\ C+(N-\epsilon\ +1)^D$

Expressed using big O notation the space complexity is: $O\left(N^D\right)$

2.5 Convolution method

We named the second algorithm to be introduced here the convolution gliding-box method. It is based on the convolution theorem, which states that the convolution of two functions equals the inverse Fourier transform of the point-wise product of the two functions' Fourier transforms. For this method, data conditioning means the generation of the Fourier transform of the original grayscale image.

2.5.1 2D algorithm

For the 2D convolution method, the 2D Fourier transform of the original grayscale image $\widehat{S}$ is created. To calculate the box masses, we need to create a kernel with the parameters of the box size to do the convolution with. Utilizing the fact that the image size is equal in both dimensions, we define the kernel E as a one-dimensional array of N length with the first ε elements being one and the rest being zero. The 1D Fourier transform of the kernel is $\widehat{E}$. Then, to get the Fourier transform of the complex box masses we multiply the elements

$$\widehat{M_c}\,(i,j)=\widehat{S}\,(i,j)\cdot\ \widehat{E}\,(i)\cdot\ \widehat{E}\,(j)\ \ , \tag{12}$$

where $\widehat{M_c}\,(i,j)$ is the element in the i-th row and j-th column in the array of the Fourier transform of the complex box masses. Then, to get the complex box masses M_c, we take the 2D inverse Fourier transform of $\widehat{M_c}$. In the final step, we calculate the amplitudes of the complex box masses to get the actual box masses M. It is important to note that in this method M, M_c and $\widehat{M_c}$ are N by N arrays, thus we need to cut out the first $N-\epsilon\ +1$ by $N-\epsilon\ +1$ part of this array to get the box masses. This process is shown in Fig. 5. The operation cost for this method using the fast Fourier transform is:

$$O_{Fourier\,2D}\,(N,\epsilon\,)\ \sim\ 4\cdot\ N^2\cdot\ log_2N+7\cdot\ N^2+N\cdot\ log_2N \tag{13}$$

The operation cost of the FFT was shown by Cochran [21]. The three parts of the operation cost in order are the 2D FFT of the image, the complex multiplication as well as amplitude calculation and the 1D FFT for the kernel.

2.5.2 3D algorithm

In the 3D version of the convolution method, data conditioning means the generation of the 3D Fourier transform of the original grayscale image $\widehat{S}$. To calculate the box masses, we need to create a kernel with the parameters of the box size to do the convolution with. Utilizing the fact that the image size is equal in both dimensions, we define kernel E as a one-dimensional array of N length with the first ε elements being one and the rest being zero. The 1D Fourier transform of the kernel is $\widehat{E}$. Then, to get the Fourier transform of the complex box masses, we multiply the elements

$$\widehat{M_c}\,(i,j,k)=\widehat{S}\,(i,j,k)\cdot\ \widehat{E}\,(i)\cdot\ \widehat{E}\,(j)\cdot\ \widehat{E}\,(k)\ \ , \tag{14}$$

where $\widehat{M_c}\,(i,j,k)$ is the element in the i-th row, j-th column and k-th page in the array of the Fourier transform of the

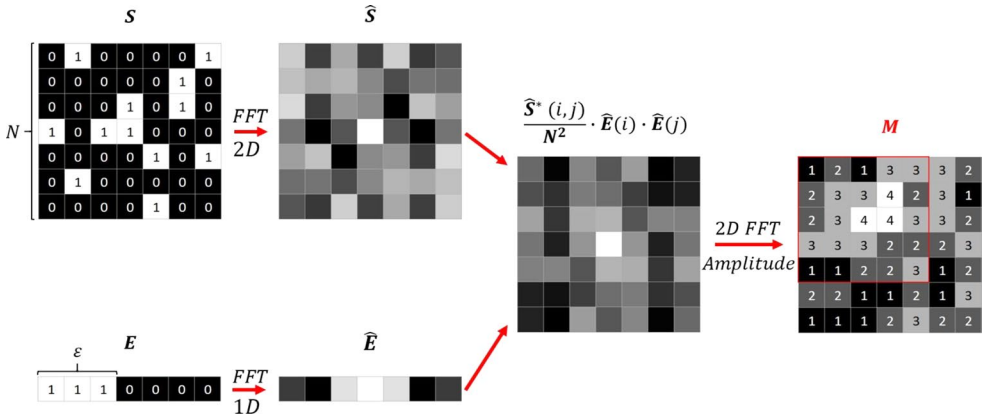

Fig. 5 Flowchart of the convolution gliding-box method. Convolution gliding-box method on an image of s $N=7$ at a box size of $\varepsilon=3$. The original image (S) as well as a kernel (E) bearing the parameters of the box size are Fourier transformed ($\widehat{S}$ and $\widehat{E}$ respectively). These Fourier transforms are multiplied as shown by the formula, where * stands for the complex conjugation. The box masses (M) are calculated by taking the top left corner of the amplitudes of the Fourier transform of the aforementioned multiplied product

complex box masses. Then, to get the complex box masses M_c, we take the 3D inverse Fourier transform of $\widehat{M_c}$. In the final step, we calculate the amplitudes of the complex box masses to get the actual box masses M. It is important to note that in this method M, M_c and $\widehat{M_c}$ are N by N by N arrays, thus we need to cut out the first $N-\epsilon+1$ by $N-\epsilon+1$ by $N-\epsilon+1$ part of this array to get the box masses. The operation cost for this method using the fast Fourier transform is:

$$O_{Fourier\,3D}(N,\epsilon) \sim 6\cdot N^3\cdot log_2 N + 9\cdot N^3 + N\cdot log_2 N \qquad (15)$$

The three parts of the operation cost in order are the 3D FFT of the image, the complex multiplication as well as amplitude calculation and the 1D FFT for the kernel.

2.5.3 Space complexity

The input of the convolution algorithm is the D dimensional Fourier transform of the original image. The auxiliary space contains the kernel and its Fourier transform, the result of the multiplication of the kernel and the input, the Fourier transform of this result and the amplitude calculation ending in the box masses.

Input space: $2N^D$
Auxiliary space:
$N + N\cdot\log_2 N + 2\cdot D\cdot N^D\cdot\log_2 N + 4N^D$
Total: $N + N\cdot log_2 N + 2\cdot D\cdot N^D\cdot log_2 N + 6N^D$

Expressed using big O notation the space complexity is: $O(N^D\cdot log_2 N)$

Table 2 Test parameters

Tested image sizes:		Tested box sizes:	
2D	3D	2D	3D
N (px):	N (px):	ε (px):	ε (px):
512	128	2	2
1024	256	4	4
2048	512	8	8
4096	800	16	16
8192	900	32	32
16,384		64	64
		128	128
		256	256
		512	512
		1024	
		2048	
		4096	
		8192	

2.6 Test methodology

To compare the performance of the five described methods, we implemented the algorithms in LabView. The codes were written to use only a single thread. We created test files to test the dependence of each algorithm on box size (ε) and image size (N), both in two and three dimensions. The only image texture dependent method we described was the trace method. To compare it with the other ones in a consistent manner we chose binary images with 25% of the pixel values being 1, and 75% of the pixel values being 0 as the test files. This means that the number of object pixels (C) is $N^2/4$. The test parameters are listed in Table 2.

All tests were run three times to measure the time needed to complete the box mass calculation. The fastest run of the three was chosen for each box size and image size. Any test that would have taken over 30 min was stopped, and the total result was extrapolated based on the number of the

calculated box sizes. The test system was an AMD Ryzen 7 5800×3D based system with 32 GBs of dual channel, dual rank, 3733 MHz tuned DDR4 memory. The full specification sheet is available in the Supplementary Materials. Along with measuring the runtimes, we calculated a theoretical runtime from the operation cost for each measurement point. To this end, we approximated how much time one operation takes by dividing the number of clock-cycles a simple double-precision floating-point addition takes (3) by the operating frequency of the CPU (4450 MHz). This results in one addition taking $6.74 \cdot 10^{-10}$ s. While not all operations are additions, these calculations can showcase how (non)optimal the codes are.

3 Results

3.1 Operation count and runtime

Calculated and measured runtimes as a function of image size for all five algorithms, both for 2D and 3D images, are shown in Fig. 6.

The calculated and measured runtimes follow similar curves in all five cases. Differences between the estimates and measurements are mostly caused by inefficiencies in the code and memory performance even though the 96 MB L3 cache of the CPU alleviates many memory issues. The largest difference was shown by the integral method, where the measured runtime was approximately threefold of the calculated runtime. All five methods scale similarly with image size. For the trace gliding-box method, scaling is caused by the fact that in the test images the number of object pixels is proportionate to their size. In terms of speed, the integral method outperformed its rivals; the convolution and the sides gliding-box methods came second and third, respectively. These three techniques were by orders of magnitude faster than the original method. The trace method did not provide better results than the original one. The integral method's runtime for the 2D image size 512 pixels at 256 box size is missing because we were unable to measure it. (The built-in timing measurement returned 0s.)

The calculated and measured runtimes as a function of box size for all five algorithms, both for 2D and 3D images, are shown in Fig. 7.

The calculated and measured results show good agreement for the various box sizes too. The differences are also similar, as they stem from the code and memory performance. Unlike in the image size comparison, in the comparison of box sizes the methods can be divided into three groups in terms of scaling. The first group contains positive scaling methods in which the runtime increases with box size. Among these methods the trace method is monotonous and will always have a higher runtime for larger box sizes. The original method has its maximum at half the image size, while the sides method has its maximum at a third of the image size for 2D images and at two fifths for 3D images. These two methods get slower for larger box sizes, up to their respective maximums, and get faster for even larger box sizes symmetrically in case of the original method and asymmetrically for the sides method. The second group contains constant methods, which have identical runtimes for all box sizes. In our case, only the convolution method belongs here. The third group contains negative scaling methods, which become faster at larger box sizes. The only such method in our study is the integral method. Table 3 shows the time and space complexity of each studied algorithm for "D" dimensional data.

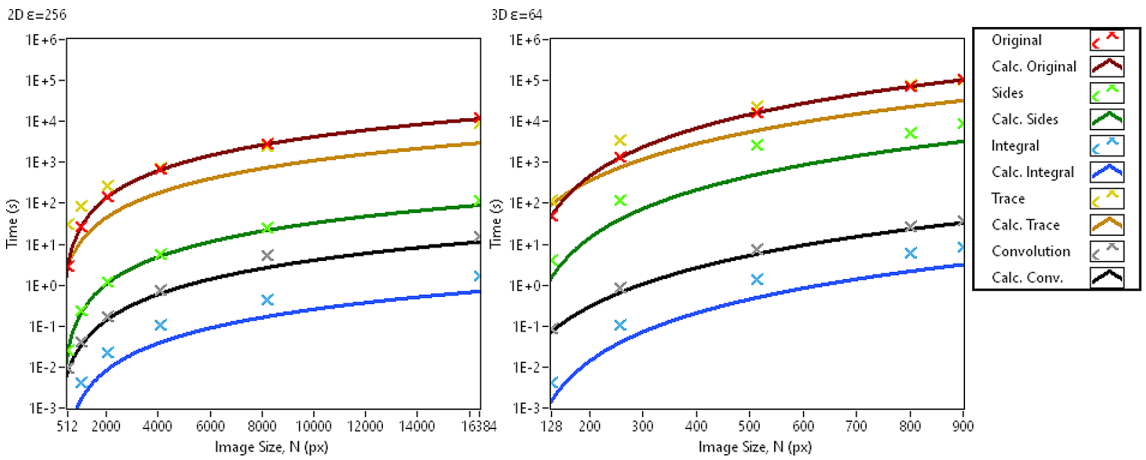

Fig. 6 Image size dependence of the runtimes. Measured and calculated runtimes of the original gliding-box method (bright red, red), the sides gliding-box method (green, dark green), the integral gliding-box method (light blue, dark blue), the trace gliding-box method (yellow, orange) and the convolution gliding-box method (gray, black) for different image sizes on 2D images at 256 box size (left) and on 3D images at 64 box size (right)

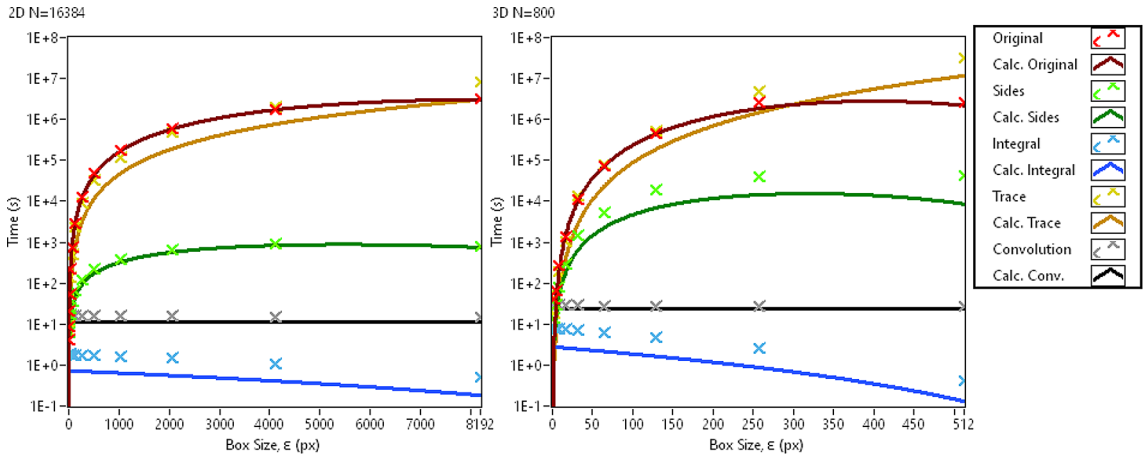

Fig. 7 Box size dependence of the runtimes. Measured and calculated runtimes of the original gliding-box method (bright red, red), the sides gliding-box method (green, dark green), the integral gliding-box method (light blue, dark blue), the trace gliding-box method (yellow, orange) and the convolution gliding-box method (gray, black) for different box sizes on 2D images at 16,384 image size (left) and on 3D images at 800 image size (right)

Table 3 Time and space complexity of the five studied algorithms described using big O notation

	Time Complexity	Space Complexity
Original	$O\left(N^D\right)$	$O\left(N^D\right)$
Sides	$O\left(N^D\right)$	$O\left(N^D\right)$
Integral	$O\left(N^D\right)$	$O\left(N^D\right)$
Trace	$O\left(C\right)$	$O\left(N^D\right)$
Convolution	$O(N^D \cdot log_2 N)$	$O(N^D \cdot log_2 N)$

3.2 Effects of multithreading

We have implemented multithreaded versions of all studied algorithms to investigate thread scaling up to 8 threads. Our findings are depicted in Fig. 8. The original algorithm showed nearly perfect scaling closely followed by the trace and sides algorithms which have only regressed in scaling when jumping to 8 threads. The integral and convolution methods have shown less than perfect scaling consistently throughout all tested thread counts.

3.3 Memory usage

Table 4 shows the maximum simultaneous memory usage and block counts reported by the Performance and Memory profiling tool in LabView for each algorithm for a 2D, 16,384 by 16,384 image at a box size of 16, and a 3D, 800 by 800 by 800 image at a box size of 8.

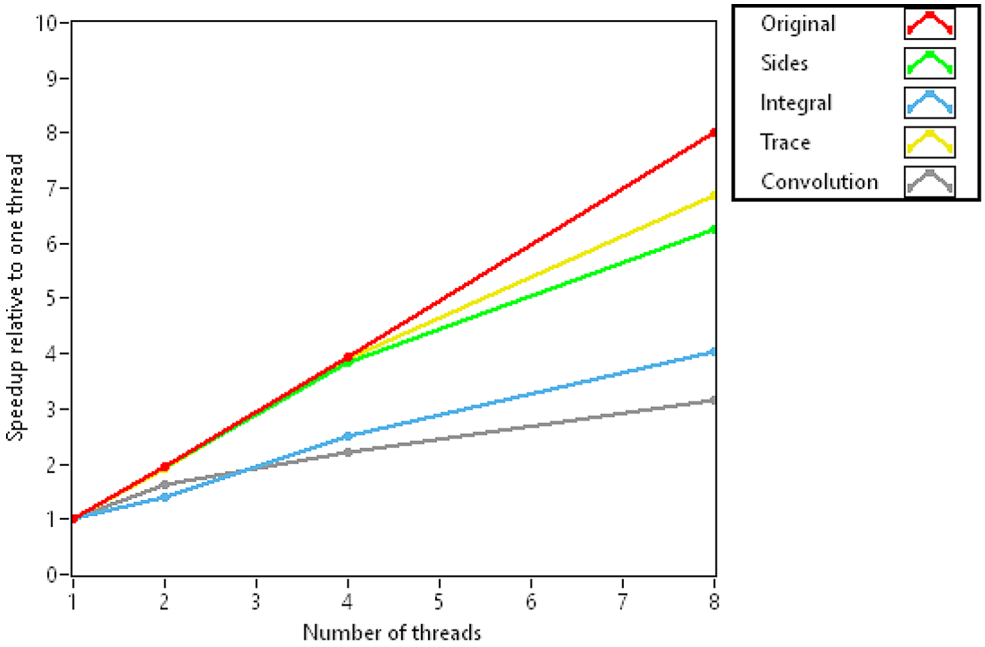

Fig. 8 Thread scaling of lacunarity calculating algorithms. Completion time of all algorithms were measured for the same test dataset utilizing 1, 2, 4, and 8 threads respectively

Table 4 Memory usage of the five studied algorithms

	2D image N: 16,384, ε: 16		3D image N: 800, ε: 8	
	Memory (MB)	Blocks	Memory (MB)	Blocks
Original	5130	30	9627	52
Sides	5131	31	9632	58
Integral	11,574	48	13,753	56
Trace	8352	32,809	16,797	4882
Convolution	18,016	51	26,016	75

The highest memory allocation was reported for the convolution method due to the use of double-precision complex floating-point numbers. The trace and integral methods showed higher memory usage than the original one. The sides method's memory usage was similar to that of the original method.

3.4 Applications

Although there are several potential applications where the newly introduced algorithms can be used effectively, here we present only one such application, in the field of optical microscopy. Single molecule localization based super-resolution microscopy (SMLM) produces images that are typically sparsely populated with object pixels. The trace method is an ideal evaluation algorithm for such images allowing excellent performance that can rival or even outperform the integral method. A typical SMLM image of

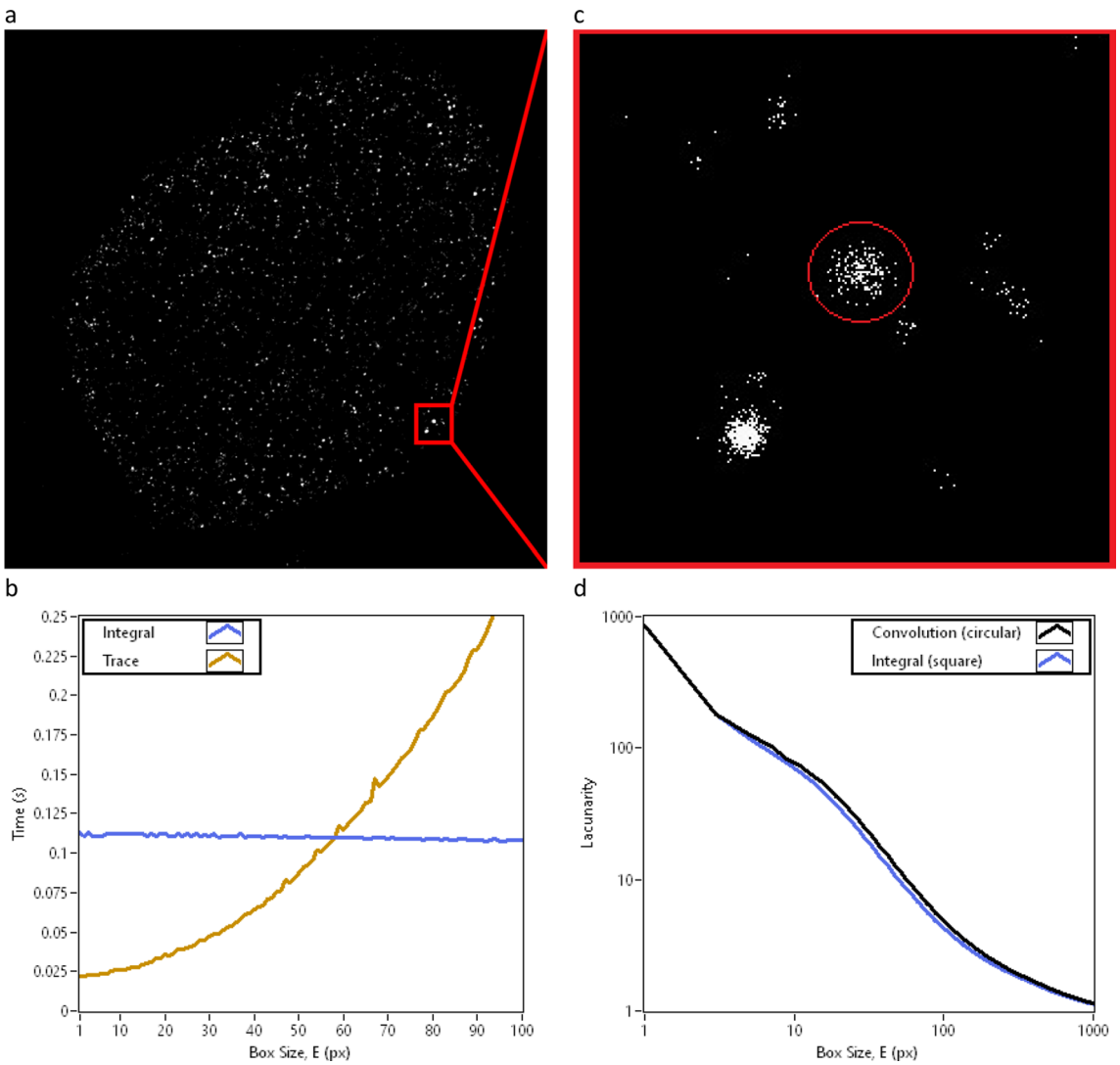

Fig. 9 Applications for newly developed lacunarity algorithms. Localization microscopy image of DNA repair foci in a U2OS cell, visualized by labeling phosphorylated H2AX (**a**) [13]. The image area is 20 by 20 μm with a pixel size of 5 nm. Runtime measurement of the trace and integral methods on the image for different box sizes (**b**). A zoomed in part of the image highlighting the circular shape of the foci (**c**). Lacunarity curves using a circular box shape with ε diameter calculated with the convolution method and a traditional ε side length square box shape calculated with the integral method (**d**)

Table 5 Representation of the strengths and weaknesses of all tested algorithms

Category	Original	Sides	Integral	Trace	Convolution
Complexity:	*Low*	*High*	*Medium*	*Low*	*Medium*
Runtime:	*Long*	*Medium*	*Short*	*Variable**	*Short*
Memory:	*Low*	*Low*	*Medium*	*Medium*	*High*
Implementation Flexibility:			*Only rectangular boxes are possible.*	*Image density dependent runtime.*	*High versatility in box shape.*

DNA repair foci of an U2OS cell nucleus can be seen in Fig. 9a [13]. The lacunarity calculation time for different box sizes are also shown in Fig. 9b. In case of small box sizes (<60 pixels) the trace method performs better than the integral approach. It is worth noting here that in SMLM this is the most interesting range of sizes. In cell biology, some subcellular structures, such as the aforementioned DNA repair foci, are close to spherical or consist of subunits that are spherical in shape as can be seen in Fig. 9c. For the geometrical analysis of such structures, a circular box shape can be advantageous as it provides higher sensitivity for textural changes on the scale of the feature size resulting in a higher lacunarity value shown in Fig. 9d.

4 Discussion

We have introduced two new gliding-box lacunarity calculating algorithms and compared them to the original method and two previously shown methods in terms of speed and memory usage for both 2D and 3D datasets. Each of the described algorithms has advantages and disadvantages that can be seen in Table 5.

The fast and accurate computation of lacunarity can be beneficial in several scientific fields. The most widely used and precise method for calculating lacunarity is the gliding-box method. As the original version of this algorithm struggles with larger datasets, we have introduced and shown several methods that can be used for larger datasets and are orders of magnitude faster than the original method. We described these methods in detail for both 2D and 3D applications making them more accessible to researchers.

Summarizing our findings, we can declare that in most cases the integral method produces the best results. It can compute lacunarity in the shortest overall timeframe without a significant increase in memory usage, which makes it an excellent candidate earning our recommendation for general use. However, it is unable to compute non-rectangular box shapes and scales inversely with box size, for in most applications lower box sizes are of higher interest. Because of these two drawbacks, for applications where rectangular boxes are not suitable, we recommend the convolution method, which was the second fastest in our tests. Because of its high memory usage, this method has its limitations too. There are several applications, such as single molecule localization microscopy, where the number of object pixels can be as low as 10% of the total pixel count or even lower. In these applications, we recommend the trace method which can provide unmatched speed for the lower box sizes on sparsely populated images. Finally, in applications where the number of object pixels is not exceptionally low and memory usage is of extreme priority, we recommend the sides method, as it needs the lowest memory allocation comparable to the original method, but is still significantly faster. Although many of these methods have shown their merit in other subfields of image processing, in the field of lacunarity analysis they have been rarely (sides method, integral method) or never (trace method, convolution method) used before. We believe our paper can popularize these more advanced methods and accelerate progress in the field of lacunarity analysis.

Acknowledgements Project no TKP2021-NVA-19 has been implemented with the support provided by the Ministry of Culture and Innovation of Hungary from the National Research, Development and Innovation Fund, financed under the TKP2021-NVA funding scheme.

Author contributions BBHK conceived the study, developed the novel algorithms, implemented the already existing algorithms, generated test data, carried out runtime measurements, compared the algorithms, created the figures, and drafted the manuscript. ME participated in the conception of the study, supervised the research, participated in data interpretation, reviewed and edited the manuscript. All authors have reviewed the manuscript and agreed to its publication.

Funding Open access funding provided by University of Szeged.

Data availability No datasets were generated or analysed during the current study.

Declarations

Competing interests The authors declare no competing interests.

References

1. Gefen Y, Meir Y, Mandelbrot BB, Aharony A (1983) Geometric implementation of hypercubic lattices with noninteger dimensionality by use of low lacunarity fractal lattices. Phys Rev Lett 50:145. https://doi.org/10.1103/PhysRevLett.50.145
2. Ivanovici M, Richard N (2009) The lacunarity of colour fractal images. In 2009 16th IEEE international conference on image processing (ICIP) (pp. 453–456). IEEE. https://doi.org/10.1109/ICIP.2009.5414394
3. Plotnick RE, Gardner RH, Hargrove WW, Prestegaard K, Perlmutter M (1996) Lacunarity analysis: a general technique for the analysis of spatial patterns. Phys Rev E 53:5461. https://doi.org/10.1103/PhysRevE.53.5461
4. Drăghici CC, Andronache I, Ahammer H, Peptenatu D, Pintilii RD, Ciobotaru AM, Simion AG, Dobrea RC, Diaconu DC, Vișan MC, Papuc RM (2017) Spatial evolution of forest areas in the northern Carpathian Mountains of Romania. Acta Montanist Slovaca, 22
5. García-Farieta JE, Casas-Miranda RA (2018) Effect of observational holes in fractal analysis of galaxy survey masks. Chaos Solitons Fractals 111:128–137. https://doi.org/10.1016/j.chaos.2018.04.018
6. Sebők D, Vásárhelyi L, Szenti I, Vajtai R, Kónya Z, Kukovecz Á (2021) Fast and accurate lacunarity calculation for large 3D micro-CT datasets. Acta Mater 214:116970. https://doi.org/10.1016/j.actamat.2021.116970
7. Waliszewski P (2016) The quantitative criteria based on the fractal dimensions, entropy, and lacunarity for the spatial distribution of cancer cell nuclei enable identification of low or high aggressive prostate carcinomas. Front Physiol 7:34. https://doi.org/10.3389/fphys.2016.00034
8. Sanghera B, Banerjee D, Khan A, Simcock I, Stirling JJ, Glynne-Jones R, Goh V (2012) Reproducibility of 2D and 3D fractal analysis techniques for the assessment of spatial heterogeneity of regional blood flow in rectal cancer. Radiology 263:865–873. https://doi.org/10.1148/radiol.12111316
9. Al-Kadi OS, Watson D (2008) Texture analysis of aggressive and nonaggressive lung tumor CE CT images. IEEE Trans Biomed Eng 55:1822–1830. https://doi.org/10.1109/TBME.2008.919735
10. Neves LA, Nascimento MZ, Oliveira DLL, Martins AS, Godoy MF, Arruda PFF, de Santi Neto D, Machado JM (2014) Multiscale lacunarity as an alternative to quantify and diagnose the behavior of prostate cancer. Expert Syst Appl 41:5017–5029. https://doi.org/10.1016/j.eswa.2014.02.048
11. Borys P, Krasowska M, Grzywna ZJ, Djamgoz MB, Mycielska ME (2008) Lacunarity as a novel measure of cancer cells behavior. BioSystems 94:276–281. https://doi.org/10.1016/j.biosystems.2008.05.036
12. Nichita MV, Paun MA, Paun VA, Paun VP (2019) Fractal analysis of brain glial cells. Fractal dimension and lacunarity. Univ Politeh Buchar Sci Bull Ser Appl Math Phys 81:273–284
13. Kovács BBH, Varga D, Sebők D, Majoros H, Polanek R, Pankotai T, Hideghéty K, Kukovecz Á, Erdélyi M (2022) Application of Lacunarity for quantification of single molecule localization Microscopy images. Cells 11:3105. https://doi.org/10.3390/cells11193105
14. Paun MA, Postolache P, Nichita MV, Paun VA, Paun VP (2023) Fractal Analysis in Pulmonary CT images of COVID-19-Infected patients. Fractal Fract 7:285. https://doi.org/10.3390/fractalfract7040285
15. Mandelbrot BB (1983) The Fractal geometry of Nature. W.H. Freeman, San Francisco, CA, USA, pp 310–319
16. Allain C, Cloitre M (1991) Characterizing the lacunarity of random and deterministic fractal sets. Phys Rev A 44(6):3552. https://doi.org/10.1103/PhysRevA.44.3552
17. Tolle CR, McJunkin TR, Gorsich DJ (2008) An efficient implementation of the gliding box lacunarity algorithm. Physica D 237:306–315. https://doi.org/10.1016/j.physd.2007.09.017
18. Backes AR (2013) A new approach to estimate lacunarity of texture images. Pattern Recognit Lett 34:1455–1461. https://doi.org/10.1016/j.patrec.2013.05.008
19. Williams DP (2015) Fast unsupervised seafloor characterization in sonar imagery using lacunarity. IEEE Trans Geosci Remote Sens 53:6022–6034. https://doi.org/10.1109/TGRS.2015.2431322
20. Viola P, Jones M (2004) Robust real-time object detection. Int J Comput Vis 57:137–154
21. Cochran WT, Cooley JW, Favin DL, Helms HD, Kaenel RA, Lang WW, Maling GC, Nelson DE, Rader CM, Welch PD (1967) What is the fast Fourier transform? Proc IEEE 55:1664–1674. https://doi.org/10.1109/MSPEC.1967.5217220