Computing and Informatics, Vol. 41, 2022, 957–980, doi: 10.31577/cai_2022_4_957

CANNY EDGE DETECTION ANALYSIS BASED ON PARALLEL ALGORITHM, CONSTRUCTED COMPLEXITY SCALE AND CUDA

Lesia I. Mochurad

Lviv Polytechnic National University
S. Bandera str., 79013, Lviv, Ukraine
e-mail: lesia.i.mochurad@lpnu.ua

Abstract. Edge detection is especially important for computer vision and generally for image processing and visual recognition. On the other hand, digital image processing is widely used in multiple science fields such as medicine, X-ray analysis, magnetic resonance tomography, computed tomography, and cosmology, i.e. information collection from satellites, its transferring, and analysis. Any step of image processing, from obtaining the image to its segmentation and object recognition is followed by image noise. The processing speed is vital in popular fields that demand image analysis in real time. In this work, we have proposed an approach of parallel computing of the Canny algorithm using CUDA technology, the complexity of object recognition is analyzed according to the type of the image noise and the level of its density. The sequenced implementation on GPU and the parallel implementation on GPU was considered. The results were analyzed for efficiency and reliability. Also, parallel acceleration is calculated according to the size of the incoming image. The manipulations with the image showed the growth of processing speed of 68 times, whereas the manipulations with the size of the kernel showed the growth of processing speed of 26 times. Another contribution of this work is the analysis of the complexity of object recognition depending on the type of image noise and the level of its density. Furthermore, the increase of Gaussian noise density linearly increases the complexity of object recognition.

Keywords: Image processing, optimization of the computing process, parallel acceleration, parallel implementation on the GPU, Gaussian noise intensity

Mathematics Subject Classification 2010: 68U10

1 INTRODUCTION

Digital image processing is widely used in various fields of science such as medicine [1] and [2] – X-ray analysis, Magnetic Resonance Imaging, Computed Tomography scan, cosmology [3] – satellite information gathering, transmission, and analysis. Noise on the image accompanies any of the stages of image processing – from their reception to segmentation and object recognition.

Pre-processing of images is an important step, as it affects the quality of the image, the accuracy, and the efficiency of the results obtained during the medium and high level of processing. It is important because the images are influenced by noise of diverse nature. If left intact, it will have a negative impact on the following levels. These noises (divided into additive Gaussian and impulse noises, which can be a component of the object under investigation) are formed when the image is transmitted via communication channels. In addition to image filtering, the brightness correction is improved. This is needed to select individual micro-objects. There are also modern image enhancement techniques such as alignment of the histograms, contrast-adaptive alignment of histograms, and Multi-Scale Retinex.

In the works [4] and [5], the authors proposed a neural network and algebraic approaches, respectively, to provide image superresolution to improve further intellectual analysis.

The field of image processing, despite its importance, has some drawbacks. In particular, when processing images, it is important to get the result in real time. However, consistent algorithms for larger images do not allow us to obtain this easily. Even when using the Canny edge detection in the field of contour detection, which dramatically reduces the amount of data that needs to be processed compared to other first-order algorithms, we need to optimize the computational process [6]. In addition, the presence of noise in images complicates their processing by both humans and computer systems [7].

Thanks to the development of the image processing site, it is now possible to use the acquired knowledge to improve it and remove interference. Noise in images occurs at different stages of their formation and use, from digital recording to user-to-user transmission. They arise for various reasons that are sometimes difficult to eliminate. Image processing makes it possible to eliminate unwanted data from an image and to increase the possibility of human perception of them in the course of their activity. Image analysis has wide application to:

- analysis and processing of two-dimensional images (photographs) in compression, coding, object recognition, improvement of image quality;
- analysis and processing of moving images during video compression, automatic quality control of vehicle tracking;
- biomedical applications – analysis and recognition of images from ultrasound diagnostics (ultrasound), magnetic resonance imaging, computed tomography scan, and other technologies.

In this paper, the Canny edge detection algorithm is investigated. As is known [8], it consists of five steps: applying a Gaussian filter, finding the intensity gradient, finding local maxima, applying a double threshold, and then tracking the edges with hysteresis. The size of the Gaussian filter and the double threshold can be manipulated to affect both the computation time and the efficiency of the algorithm. Based on the images being processed, different combinations of these two parameters can give more "accurate" results.

Detecting contours using a sequential algorithm leads to slow operation, especially when the number of pixels increases because the amount of work is directly proportional to the number of pixels in the image. For real-time image processing analysis, sequential execution will be too slow. It is important to offer an approach to parallelization of a sequential algorithm and to study how the computation time changes with increasing image size. Parallelization in the work is carried out on the basis of CUDA technology. It allows programmers to write thread-level code that runs on many independent threads at the same time. This model is ideal for parallel data processing. For example, it allows you to apply the same filter to many pixels at once.

The paper also analyzes the complexity of object recognition depending on the type of noise and the value of noise intensity. It is important to obtain information about the dependence of the difficulty of detecting objects in the image on the noise component of the image. Such information can be useful for assessing the feasibility of using an image for a particular field of activity, given the nature of the noise in the image.

The main contribution of this paper can be summarized as follows:

1. This paper designs a computationally efficient approach to edge detection of the image based on a Canny algorithm and the CUDA parallel computing technology. It provides the possibility of significantly optimizing the computing process by its parallelization; the ability to solve the task for the case of large image processing.

2. It analyzes the complexity of object recognition depending on the type of noise and the value of noise intensity. A scale of complexity of recognition of objects of this class on the image depending on noise is constructed for the first time.

3. It carries out the experimental investigation of the efficiency of work of all proposed algorithms; it receives an acceleration about 68 times and 26 when changing the image size and kernel, respectively.

The rest of the paper is organized as follows: The analysis of literary sources is presented in Section 2. Descriptions of Canny's algorithm and designed approaches are shown in Section 3. Section 4 comprises the results of numerical experiments of the proposed algorithmic implementation of the developed method. Conclusions and prospects for further research are presented in the last section.

2 LITERATURE REVIEW

There are various methods for detecting the edges of images, among which include:

Sobel's operator [9]. One way to find boundaries in an image is to implement a Sobel filter or operator, which allows you to find an inaccurate approximation of the brightness gradient of the image. Strictly speaking, the operator uses 3×3 cores, with which the original image is collapsed to calculate the approximate values of the derivatives horizontally and vertically.

The Sobel edge detector uses two 3×3 cores, one estimates the gradient in the x-direction, and the other estimates the gradient in the y-direction. The kernel passes through the image, manipulating one square of pixels at a time. The algorithm calculates the gradient of the image intensity at each point and then gives the direction to increase the image intensity at each point from light to dark. Areas of the edges represent contrasts of strong intensity, which are darker or brighter. Sobel's algorithms work using a mathematical procedure called convolution, and usually analyze the derivatives or second derivatives of digital numbers in space.

The Prewitt operator [10]. A method of selecting boundaries in image processing that calculates the maximum response on a plurality of convolution cores to find the local orientation of the border in each pixel. It was created by Dr. Judith Prewitt to identify the boundaries of medical images. The operator uses two 3×3 cores, collapsing the original image to calculate the approximate values of the derivatives: one horizontally and one vertically.

Canny's edge detection algorithm is considered to be a better contour detection algorithm than the ones we mentioned above. This is mainly due to two steps: finding local maxima and contour tracking hysteresis. These two steps reduce the number of "false" edges and, therefore, create a better starting point for further process [11].

Simulation of different types of noise allows you to get a distorted image, based on which you can develop methods to prevent noise and improve image quality when really needed. Among the most common noise models are the following: Gaussian, Poisson models, pulse models, and others.

The research [12] proposed a model for crowd control with a focus on mask detection and people counting for the specific usage of pandemic situations. It was shown and verified that by using a deep artificial neural network, the system could perform the assigned task successfully. However, the authors do not study the time spent on identification and do not analyze the possibilities of optimizing calculations in order to make real-time detection.

In the work [13], the authors have demonstrated a version of the complete Canny edge detector under CUDA, including all stages of the algorithms. For testing, in this paper authors compared GPU Canny implementation with an assembly optimized CPU implementation in the OpenCV library, as well as with the Matlab

toolbox implementation. As a result, the authors managed to achieve a maximum acceleration of only about 4. This result is significantly improved in this paper, it received an acceleration of about 68 times and 26 when changing the image size and kernel, respectively.

In the paper [6], the authors have implemented the Canny edge detection using CUDA. In this paper, there are no results on the change in acceleration when manipulating the size of the image and the kernel. CUDA implementation achieved approximately 50 times speedup. If the data transfer time is not included, CUDA implementation achieved approximately 61 times speedup. In this paper, this result has improved by 36 %.

In the paper [14], the authors proposed to use OpenMP technology to parallelize the sequential algorithm, which is especially relevant when using the multi-core architecture of modern computers. However, significant benefits can be gained by using GPUs and CUDA technology.

There is a brief discussion about the necessity of computing GPU CUDA in the analysis of medical images in work [15]. The GPU performances of existing algorithms are analyzed and the computational gain is discussed. But here, in contrast to the proposed approach, there is no analysis of performance indicators when using CUDA technology in the process of processing images of different dimensions on GPUs.

There are some proposals for the implementation of several medical image segmentation algorithms using CUDA-supported graphics processors, comparing their performance and results with the previous implementation in the old version of the GPU and CPU, that points to the advantages of using CUDA technology and how to design algorithms for most powerful use in work [16]. In our work, the acceleration has been obtained 10 times faster for processing images of the same size. However, the paper does not provide an algorithm for eliminating the noise of different nature.

The work [17] defines a set of criteria for the effective use of graphics processors, and each segmentation method is evaluated accordingly. In addition, links to relevant GPU implementations and an understanding of GPU optimization are offered and discussed. A review of this work suggests that most segmentation techniques can benefit from the graphics processor processing through the parallel structure of the techniques and a large number of threads. However, factors such as synchronization, branch divergence, and memory usage may limit acceleration. But this work does not provide quantitative indicators of the efficiency of parallel algorithms. The proposed approach allowed to optimize the computational process by combining all filters with threads so that no two filters were applied simultaneously. This allowed us to ensure accuracy and consistency in all execution cycles.

There are many free and paid image processing software platforms that handle image processing, which also includes processing noise. Such applications are effective precisely for image processing and reduction of the noise level. However, they do not allow for analysis of the images provided and to assess the complexity of the rosetopinated objects on it. The most popular solutions are:

Adobe Photoshop – considered the most powerful tool for working with digital images. It makes it possible to clean images from noise in various ways, including using plugins. This software solution allows you to apply cleaning algorithms, set your own parameters, and determine the level of noise and light. Adobe Photoshop allows you to use a variety of technologies and tools for image processing, including the processing of noise. However, it does not contain sufficient information about the noise level and does not allow for the estimation of the distortion. Therefore, this product is useful for image processing but is not effective for their analysis.

ERDAS Imagine – raster graphics editor and software product, primarily intended for remote sensing data processing. It allows to process, display, and prepare raster images for further processing. This software solution allows the processing of cartographic data. An example of image analysis is the development of a spatial model for data processing, orthotransformation, the creation of a mosaic from images, and the production of stereo images.

MacSADIE – an application for Macintosh, allows the process of both color, and grayscale images for scientific, engineering and research work. It contains about 50 commands for image analysis, spatial filtration, multispectral processing, and classification. MacSADIE offers a wide range of options for the analysis and processing of digital images.

Described software solutions allow us to perform a wide range of image processing operations and, in particular, control noise suppression. The research carried out in the presented paper provides information about the image and the complexity of human perception depending on the type and intensity of the noise. Such information can be useful in various applications where people are confronted with digital imaging.

It should also be noted that a system that analyzed the complexity of object recognition according to noise intensity was not found.

3 MATERIALS AND METHODS

The following distribution density describes the Gaussian noise model:

$$\rho(x) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

where x – brightness value, μ – the average value of a random variable, σ – standard deviation [18].

Gaussian noise most often appears on digital images during the image acquisition phase. This may be sensor noise caused by low light or high temperatures.

Impulse noise is expressed in the form of black and white dots corresponding to the places of loss of information in the transmission channels. If impulse noise is bipolar, that is, elements of both maximum and minimum intensity are present,

and especially if they have almost equal probability, it is similar to salt and pepper grains scattered over the image. For this reason, impulse interference is also called noise – "Salt and Pepper" [19].

The study of the complexity of recognizing objects of a given class in the image is based on the following input data:

- kind of noise;
- noise intensity;
- the number of recognized objects in the image.

The type of noise may be subject to the Gaussian distribution law [20].

3.1 Review of Canny's Algorithm

Before discussing how an algorithm can be optimized, it is important to understand how the algorithm is applied to the image. The Canny edge detector contains five filters that are applied sequentially to the pixel by pixel image. Because pixels have red, green, and blue components, each field is manipulated independently. Some filters require the exchange of information between pixels, while other filters are applied only to each pixel separately. The exchange of information between pixels can significantly reduce performance when implementing a parallel version of the contour detection algorithm, so the transfer of information is extremely important [6, 8].

Gaussian erosion. The first filter needed in Canny's edge detection algorithm is Gaussian blur. The purpose of this filter is to slightly blur the image and reduce the noise in it. Noise reduction in the image allows you to detect actual, clear edges, avoiding false noise, which could be defined as a false edge. This filter works by applying a square "convolution kernel" to each pixel of the image by the formula:

$$G(x,y) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}$$

where (x, y) – the pixel coordinates of the image and σ – the noise mask.

After the kernel is applied, the sum of the scalars multiplied by the pixel values inside the kernel is divided by the sum of the scalars in the convolution kernel, and this becomes the new value for pixel blur. For example, if a 5x5 convolution kernel is applied to a pixel in the center of the image, the new red, green, and blue values for that pixel will be the sum of the red, green, and blue values multiplied by the convolution kernel and divided by the sum of the convolution kernel values, respectively. What makes this Gaussian filter is that the "convolution kernel" does not follow the mean distribution, where all scalars make the same contribution, the scalars in the matrix represent the Gaussian distribution, i.e. the pixel values closer to the blurred pixel have a greater impact. The size of the convolution kernel determines the blurring of the image. The larger the kernel, the more the image is blurred.

Intensity gradient. The input to the intensity gradient is a blurred image obtained by the Gaussian blur described above. This filter takes the first derivative of the contrasts between the color channels throughout the image in the horizontal and vertical directions to determine the magnitude and direction of each possible contour. Horizontal and vertical gradients are known as "deltaX" and "deltaY" arrays.

These derivatives are converted from a three-channel gradient of red, green, and blue to a single single-channel gradient, by converting the gradients to shades of gray. The contour gradient is determined by finding the hypotenuse deltaX and deltaY. The size of the gradient can be determined using the following formula: sqrt(deltaX2 + deltaY2). The direction of the contour is evaluated as "alpha", which is the ratio of deltaX and deltaY. "Alpha" is used in the following filter [8].

Search for local maxima. Because the image was blurred during the Gaussian blur phase, the edges detected by the intensity gradient filter may appear thicker than they actually are. The search for local maxima dilutes the edges that were formed by the horizontal and vertical gradients formed by the intensity gradient filter. Because the image was blurred, the pixel values to the outside of the path are smaller than the pixels in the middle. The maximum search filter takes pixels along the edge and "mutes" values that are not the maximum value to smooth the edges. This is achieved by first determining the direction of the edge using "alpha". Pixels perpendicular to the edge marked with "alpha" are compared, and all gradient values are reduced to 0, except for local maxima, which indicates the location with the sharpest change in intensity.

Double threshold. The previous three filters have been applied to the image to detect true contours, but there may still be noise or false contours at the edges. The double threshold filter uses high and low thresholds throughout the image to identify strong and weak contours. If the pixel value exceeds a high threshold, it is denoted as a strong edge, if the pixel value is less than the high threshold value, but lower than the lower threshold value, it is denoted as a weak edge, and finally, if the pixel value is less than the low threshold, the pixel is completely suppressed. This filter is used to eliminate the last bit of noise [6].

Edge tracking hysteresis. In addition to edge detection, all four previous filters filter out the noise present in the image. Because there may be some contours that have been over-filtered (for example, edge information may be blurred in the image due to Gaussian blur), the contour tracking hysteresis filter makes one final pass over the image and connects the edges that should have been with unity. Using the strong edges that have been identified by the double threshold filter, the hysteresis contour tracking filter passes through the image using a 3×3 matrix that connects the weak contours adjacent to the strong ones. This restores the edges that should have been detected but were discarded by aggressive filtering. At the end of this filter, the edges of the image are detected properly.

3.2 Implementation of a Sequential Algorithm

To implement the algorithms CUDA platform was installed and used along with the Nvidia GeForce GTX 1050 TI graphic card. Image-processing library Magick++ was used to manipulate pixels, the library lets you import and arrange common file formats images such as .jpg, .png, or .bmp into a flat buffer of pixels.

Sequential implementation of each filter used one for loop to iterate the pixel buffer obtained with Magick++. Each Canny algorithm filter was implemented in a separate function, each function used a nested for loop to sequentially iterate through the pixel buffer and convert the pixels one by one.

Sequential implementation of these filters has made debugging algorithms much easier because it is much easier than parallel. However, this is a rather limited implementation, in terms of image size, because the processing time is linearly scaled with its size. A faster approach to image processing was required to model the analysis in real time.

3.3 Implementation of a Parallel Algorithm

A lot of tools (libraries, devices, etc.) for parallel computing that Canny edge detector consist of exist, however, API CUDA NVIDIA was used for this project. Graphics processing units are a great choice for image processing and filtration. CUDA provides the ability to write threaded code which compiles in multiple independent threads simultaneously. This technique is perfectly suitable for data parallel computing, for instance, it lets you use the same filter on multiple pixels at the same time.

The source code is written in C++ with CUDA extension and compiled with NVIDIA NVCC. To achieve API CUDA data parallelism every filter should be implemented in CUDA constructions. NVIDIA graphic cards have numerous abstraction levels which can be used for achieving better parallel computing. Any task, that should be parallely computed, can be divided into a combination of grids, blocks and threads.

The CUDA software model is presented in Figure 1. CUDA C++ extends C++, allowing the programmer to define C++ functions called kernels, which are executed N times in parallel with N different CUDA threads when called, as opposed to only once, like regular C++ functions. The kernel is determined by the *_global_* ad specifier, and the number of CUDA threads that execute that kernel for a given kernel call is determined by the new runtime configuration syntax <<< ... >>>. Each thread that executes the kernel is assigned a unique thread identifier. which is available in the kernel through built-in variables (threadIdx).

Concurrency in CUDA allows you to "unwind" for loops by matching pixels to threads [13]. If the thread can be mapped to each individual pixel, it works as if the for-loop applies the same filters to each pixel; the difference is that all pixels are calculated in parallel. The threads work together exchanging data with each other through shared memory also synchronizing their work for coordinated access to the

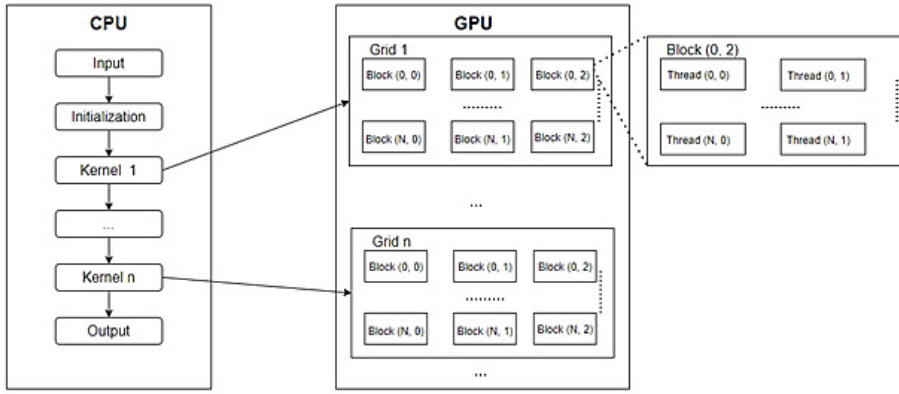

Figure 1. CUDA software model.

memory. To be more accurate, you are able to set specific points of synchronization in the kernel by calling an internal function *_syncthreads*(); *_syncthreads*() acts like a barrier that holds all the threads in the blocks until it is allowed to continue the operation. Additionally, atomic functions are used in *_syncthreads*().

The Canny edge detection algorithm problem can use domain decomposition into groups of pixels for each thread. As you know, CUDA abstracts GPU hardware resources into a grid of threading blocks, each of which can be referenced using a global thread identifier. Thread identifiers consist of one-, two-, or three-dimensional coordinates. The version of CUDA used provided a maximum of 1 024 threads in the block. The image size was divided by 1024 to get the number of blocks in the thread, and all these blocks were within one grid. One thread per pixel was efficiently started during each kernel execution, each performing the work of an internal sequential implementation cycle. A separate kernel was created for each step of the algorithm, except for the hysteresis step.

To allow parallelization of the loop tracking hysteresis filter by CUDA, the sequential implementation was divided into two parts (due to data dependency), each of which was implemented as a separate CUDA core. The first CUDA kernel passes through the image and pushes any pixel that exceeds a high threshold to its maximum value (the strong edge is equivalent to the maximum pixel value or "white"). This step also creates a bit mask of these strong edges that need to be reviewed for further processing. The second CUDA core examines each pixel that was previously a strong edge and applies a low threshold to its eight immediate neighbors. If any of these neighbors are above a low threshold, they are also referred to as strong edges (effectively connecting weak edges). This led to a slightly different contour detection, which can be seen in Section 4.

Within the NVIDIA video card memory hierarchy, Shared Memory can be used to quickly transfer information between threads [21, 22]. This memory is local to each block, so Shared Memory cannot be shared between blocks without explicitly

reading and writing to global memory (which is available to all threads, blocks, and grids). The convolution kernel used in Gaussian blur, and any other kernels or variables applied to the entire image, have also been moved to Shared Memory because it is much faster than using global memory [23, 24].

The last optimization that was used in the parallel implementation of the Canny edge detection algorithm was flows. Threads allow you to synchronize the task layer, forcing each CUDA kernel to perform tasks (kernels and memory operations) sequentially in the thread. All filters used in the algorithm were linked together by a stream, so that no two filters were applied simultaneously. This allowed us to ensure correctness and consistency in all execution cycles.

4 RESEARCH RESULTS

Figure 2 shows the original image and the results after sequential and parallel edge detection algorithms, respectively. Similarly, Figure 3 presents a reference image. Figures 4 and 5 are the results of our sequential and parallel edge detection algorithms, respectively.

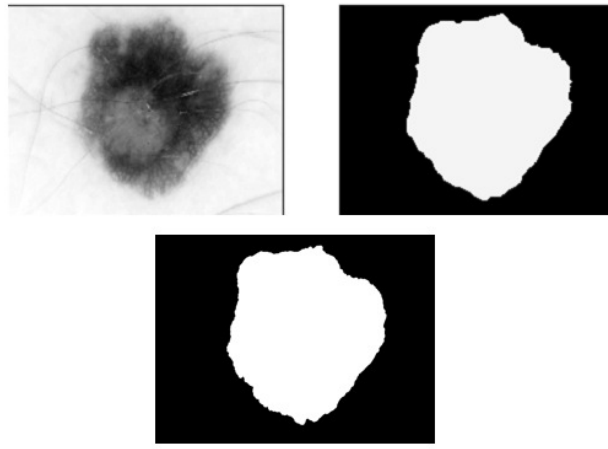

Figure 2. Original image, the results of the sequential and parallel edge detection algorithms, respectively

Based on Figures 2, 3, 4, 5 we can conclude about the reliability of the results. To test the effectiveness of parallelization, two analyzes were performed: image size manipulation and kernel size manipulation. The number of threads was not manipulated, because we coupled each thread with each pixel of the image, so as the image size increased, the number of threads increased. For image size experiments, a very small image (70×70), a medium image (640×480), a 1 080p image ($1\,920 \times 1\,080$), a 4k image ($4\,096 \times 2\,160$), an 8k image ($7\,680 \times 4\,320$) and an image ($10\,240 \times 10\,240$) were used to process the code sequentially and in parallel.

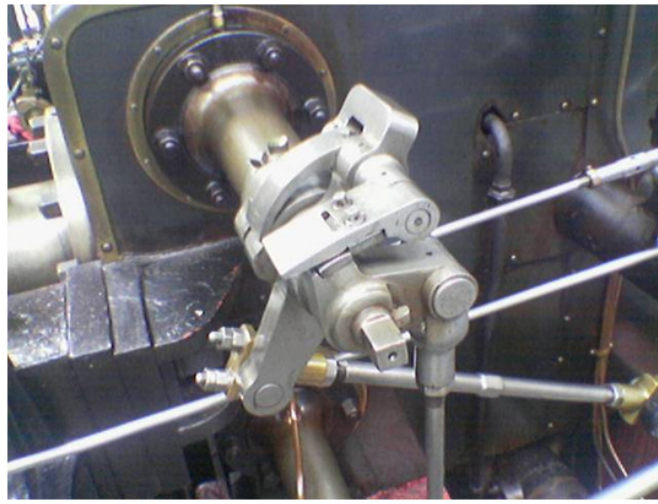

Figure 3. The original image

The time and relative acceleration [22] between sequential and parallel implementations were recorded and calculated, respectively. For experiments with convolution kernel size, the kernel size was changed from 3×3, 5×5, 7×7, 15×15, and 31×31. The 15×15 and 31×31 convolution kernels are too large to provide any proper edge detection because they blur the image too much, but the acceleration when calculating this data was still worth checking out.

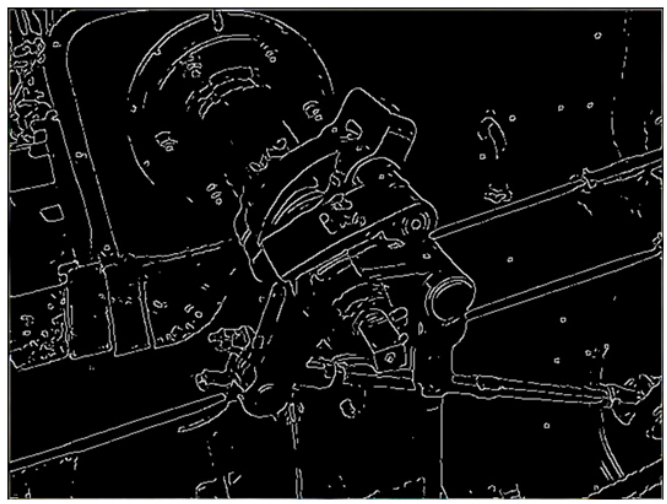

Figure 4. Images after performing a sequential algorithm

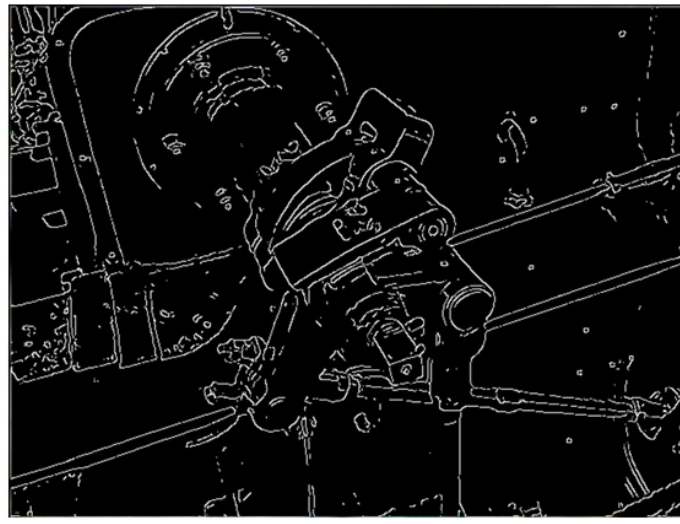

Figure 5. Image after performing a parallel algorithm

Table 1 and Figure 6 show the execution time of sequential and parallel algorithms for different image sizes. Also, the paper calculates one of the indicators of the effectiveness of the proposed approach – acceleration $S_p = \frac{t_s(s)}{t_p(s)}$, where $t_s(s)$ and $t_p(s)$ – execution time of sequential and parallel algorithms, respectively. The values of which for different image sizes are given in Table 2.

Image Extension	$t_s(s)$	$t_p(s)$
70×70	0.003811	0.086464
640×480	0.270741	0.10651
$1\,920 \times 1\,080$	1.642513	0.16379
$4\,096 \times 2\,160$	7.332309	0.405638
$7\,680 \times 4\,320$	26.205247	1.233142
$10\,240 \times 10\,240$	111.740554	1.643233

Table 1. Execution time of sequential and parallel algorithms at different image sizes

As you can see from the results, when increasing the image size there is a need to parallelize the sequential algorithm in order to obtain the result in real time. When the image size is $1\,920 \times 1\,080$, an acceleration of about 10 is achieved, at $4\,096 \times 2\,160$ – 18, at $7\,680 \times 4\,320$ – 21 and at $10\,240 \times 10\,240$ – 68. As a result, we improved the result presented in [6] by 36 %.

Table 3 and Figure 7 show the execution time of sequential and parallel algorithms at different sizes of the convolution core. The obtained acceleration is presented in Table 2 and visualized in Figure 8.

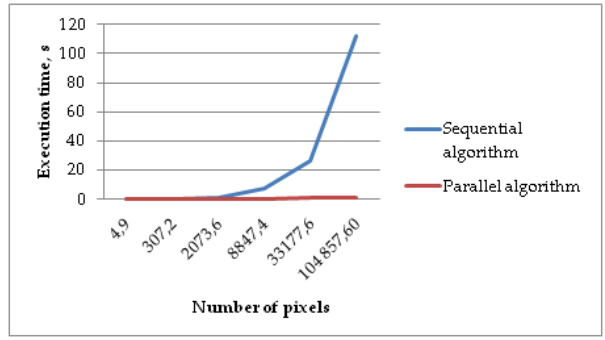

Figure 6. Comparison of execution time of sequential and parallel algorithms depending on the image size

Image Extension	S_p
70×70	0.044076147
640×480	2.541930335
$1\,920 \times 1\,080$	10.02816411
$4\,096 \times 2\,160$	18.0759914
$7\,680 \times 4\,320$	21.25079431
$10\,240 \times 10\,240$	68.0004321

Table 2. Acceleration when manipulating the image size

The Size of the Kernel	$t_s(s)$	$t_p(s)$
3	0.439012	0.118802
5	0.937175	0.136428
7	1.642513	0.16379
15	8.013535	0.418164
31	33.578706	1.28104

Table 3. Execution time of sequential and parallel algorithms at different kernel sizes

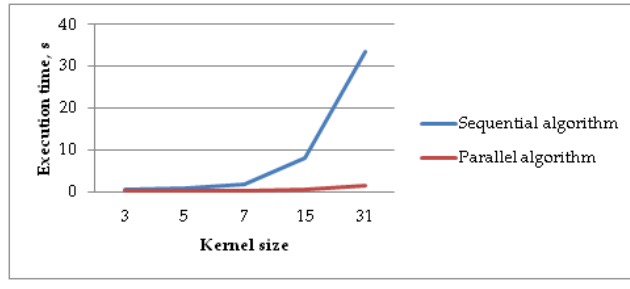

Figure 7. Comparison of execution time of sequential and parallel algorithm depending on the size of the convolution kernel

The Size of the Core	S_p
3	3.695324995
5	6.869374322
7	10.02816411
15	19.16361762
31	26.21206676

Table 4. Acceleration when manipulating the size of the convolution kernel

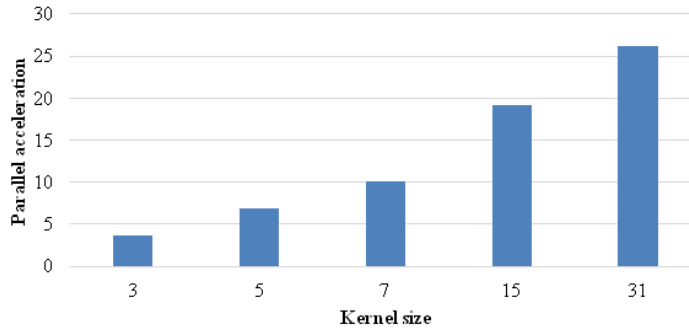

Figure 8. Dependence of parallel acceleration on the size of the kernels

As we can see, when the core size is 3, we get an acceleration of about 3.7; and with a core size of 31 – 26. This indicates the relevance of the proposed approach.

The paper also forms the methods of collecting information. For this purpose, a graphical user interface has been developed, which will allow to accumulate information for further analysis by passing tests. This information includes the number of objects detected in the image over the duration of the test using the selected noise type and intensity.

The study of the complexity of recognizing objects of a given class in the image is based on the following input data: the type of noise; noise intensity; the number of recognized objects in the image. The type of noise can be subject to the Gaussian distribution law or impulse noise. Noise intensity depends on the type of noise. If the noise obeys the Gaussian distribution law, the intensity is determined by the standard deviation – the greater the standard deviation, the denser the noise in the image. The value of the number of recognized objects in the image is determined by a human test. During the test, a person looks for objects in the image. This value is used to calculate numerical complexity as the average of the number of objects found.

Implemented user interface when starting the program has the following form:

1. Setting noise parameters:
   - choice of noise type (see Figure 9);

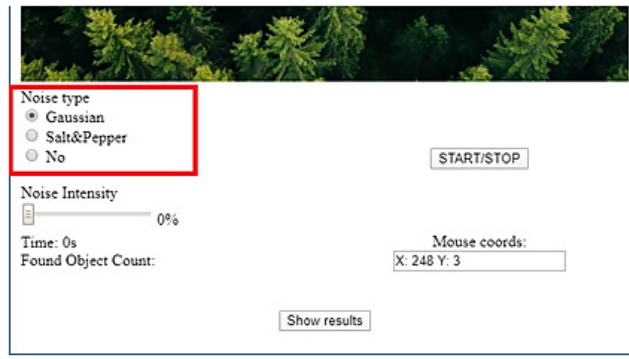

Figure 9. Noise type setting

   - choice of noise intensity (see Figure 10).

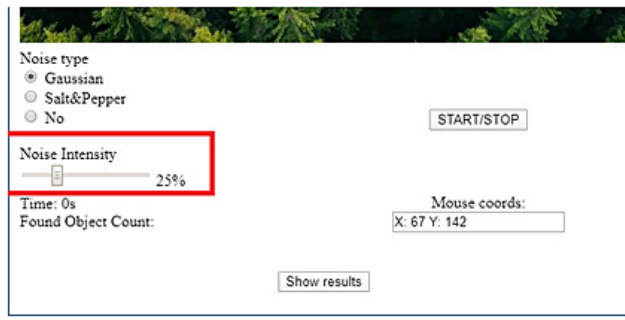

Figure 10. Setting the noise intensity

2. Passing the test with the image of the test duration and the number of recognized objects (see Figure 11);
3. Display the accumulated results through Microsoft Excel.

As a result of adding Gaussian noise to the image with a mean of 0 and a deviation of 100, the following result was obtained (see Figure 12).

Gaussian noise in the images is manifested in the form of points of random intensity, located over the entire area.

After analyzing the obtained histograms, we can draw the following conclusions. Gaussian noise shifts the histogram of the image in accordance with the distribution

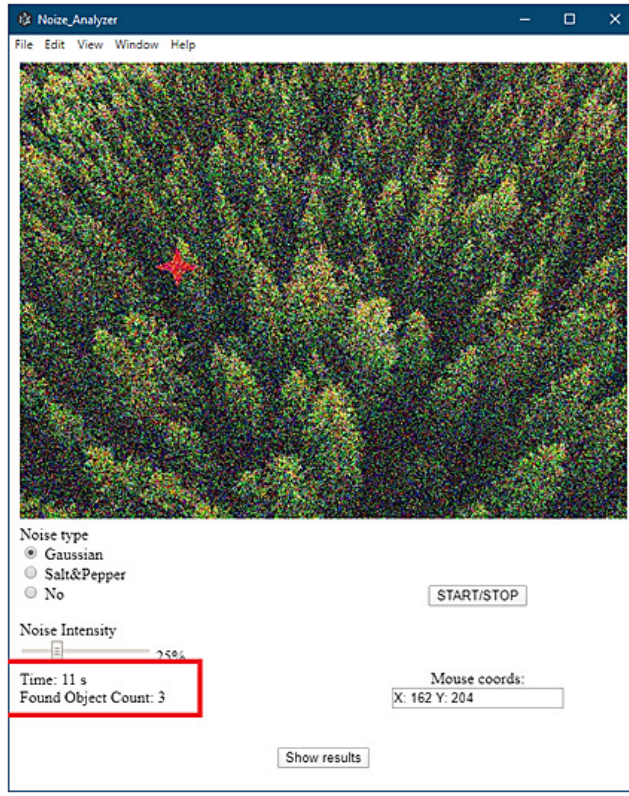

Figure 11. Passing the test

law. All three image channels were processed independently, resulting in interference of different colors. The intensity of most pixels in the original image was close to zero, and in the case of adding noise according to Gauss's law – increased for all channels.

Applying pulsed noise to the original image, we obtain the following distribution of pixel intensities (see Figure13).

Based on the analysis of the received histograms, the following conclusions can be drawn. Impulse noise leaves the same type of histogram, but adds random peaks to the image. The value of some points becomes either maximum intensity or minimal, which can be easily seen on the histogram.

Consider the initial data generated based on the tests for Gaussian noise. Table 5 shows the calculated relative complexity for each intensity level, where K – the average number of recognized objects. To calculate this value, the average number of recognized objects is first calculated and then this value is normalized by dividing by the maximum. After that, the relative complexity is calculated by the

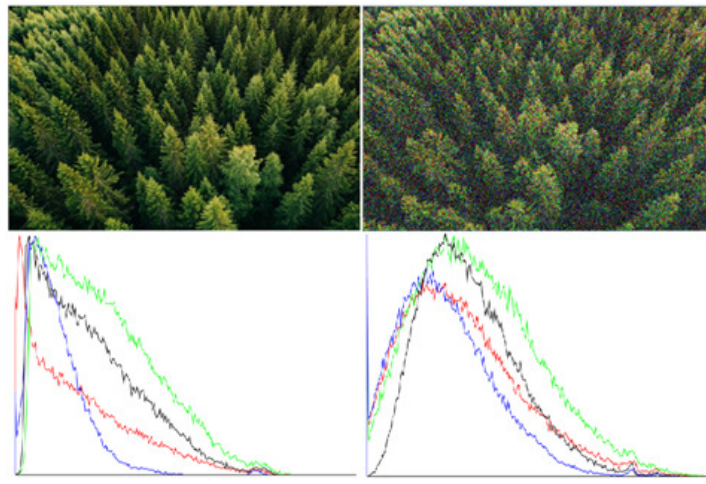

Figure 12. Histogram image with superimposed Gaussian noise

formula:

$$\text{Relative complexity} = (1 - \text{Normalized value}) * 100\,\%.$$

According to the results of the study, it can be concluded that the image with the highest value of Gaussian noise intensity has a recognition difficulty of 55.21 %, i.e., objects in such an image are the most difficult to detect by the human eye. For Gaussian noise, the intensity of 100 % corresponds to the standard deviation value

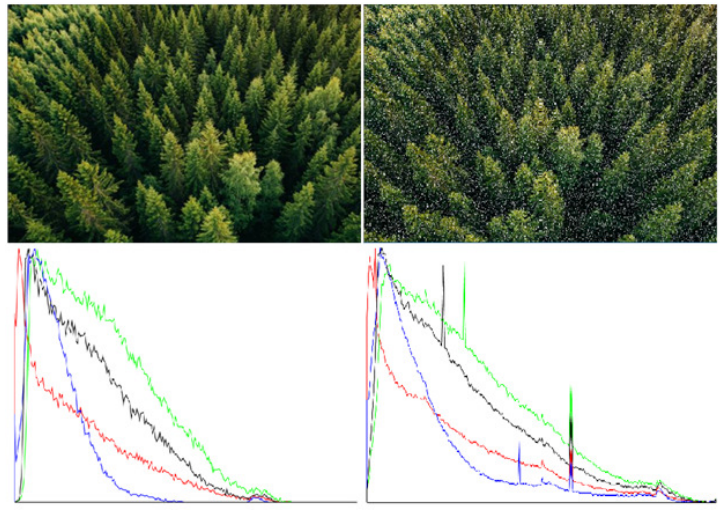

Figure 13. Histogram of the image with superimposed impulse noise

Intensity (%)	K	Normalized Value	Relative Complexity (%)
0	21.1	1.00	0.00
25	17.8	0.84	15.64
50	15.26	0.72	27.66
75	11.2	0.53	46.92
100	9.45	0.45	55.21

Table 5. The results of the assessment of complexity for Gaussian noise

of 200. From the given table it can be seen that the number of recognized objects decreases with the increasing noise intensity.

Based on the data presented in the table, you can build a scale of difficulty in recognizing objects of a given class on the image depending on the noise. Constructed scale given in Figure 14.

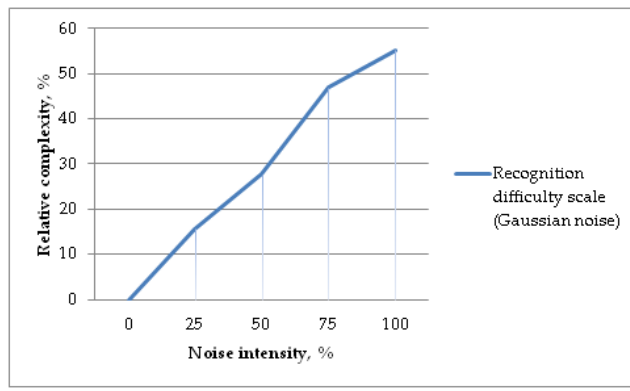

Figure 14. Recognition difficulty scale (Gaussian noise)

The results of the study on the noise model "Salt and Pepper" are presented in Table 6.

Intensity (%)	K	Normalized Value	Relative Complexity (%)
0	21.1	1.00	0.00
25	20.4	0.97	3.32
50	14.43	0.68	31.62
75	14.70588235	0.70	30.30
100	11.92857143	0.57	43.47

Table 6. Results of the assessment of complexity for noise "Salt and Pepper"

The research carried out on the noise model "Salt and Pepper" shows that for the highest intensity with the value (100 %) the relative complexity is 43.47 %. The most difficult image to recognize is one with high noise intensity. A slight variation

in the relative complexity was observed for noise intensity with a value (60 % and 80 %), which may be due to different numbers of tests passed.

According to the results of the study, a scale was constructed, which is presented in Figure 15.

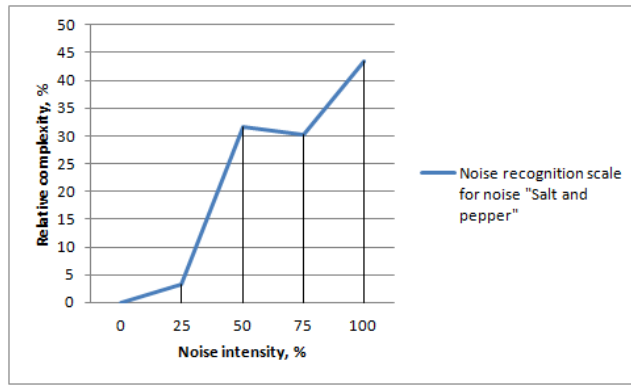

Figure 15. Recognition scale for noise “Salt and Pepper”

Comparison of the relative complexity between different types of noise (Gaussian and “Salt and Pepper”) are presented in Table 7.

Intensity (%)	Gaussian Noise (%)	Noise “Salt and Pepper” (%)
0	0	0
25	15.64	3.32
50	27.66	31.62
75	46.92	30.30
100	55.21	43.47

Table 7. Comparison of Gaussian noise and noise “Salt and Pepper”

For Gaussian noise, we can see a more straight growth of relative intensity than for the “Salt and Pepper” noise. According to the largest noise intensity Gaussian model shows a high complexity of object recognition than the noise “Salt and Pepper”. This result can be justified by the specificity of the noise model itself: Gaussian noise is applied to all 3 channels of the image, causing “mashing” (decrease of brightness) of the recognized object, while the noise model “Salt and Pepper” does not affect the brightness of the object recognized, but it can only show “drop-out” pixels.

The above results may be relevant to areas of direct human interaction with digital images, such as medical fields that use digital images. The obtained conclusions can be used to calculate the possibility of error and the relationship between image noise and the magnitude of the error.

5 CONCLUSIONS

From the above series of numerical experiments, it can be concluded that the speed of the parallel algorithm is impressive. Thus, Table 3 and Table 4 show that, at least, the proposed parallel algorithm allowed to obtain an acceleration of approximately 2.5 times in terms of image size; and almost 4 times when changing the size of the convolution kernel. At best, image size manipulation showed a 68-fold increase in speed, while manipulation of the convolution kernel size showed a speed increase of up to 26 times. As can be seen in Figure 6 and Figure 7, if larger images and kernel sizes are used, this trend also persists as the sequential implementation time increases linearly with the image/kernel size, while the parallel implementation time increases much more slowly.

The only time this did not happen was when processing the smallest image of 70×70 pixels. This image was small enough to make sequential implementation faster, at the expense of additional device memory allocation, data transfer, and bus transfer between the CPU and GPU.

The execution time of the parallel implementation remained relatively constant compared to the sequential implementation, the time of which increased linearly with the number of pixels to be processed. This is the best scenario for parallel acceleration: the execution time is almost independent of the number of iterations of processing that need to be performed on pixels. This suggests that Canny's edge detection algorithm parallels extremely well.

Noise in the image can be damaging in various industries because the image is a widely used means of transmission. This is why an analysis of possible errors in image processing is useful. The work compared the difficulty of recognizing objects in an image depending on the type of noise and its intensity. Thus, the Gaussian noise model is the most similar to the images distorted at the registration stage in case of sensor failures (for example, overheating), and the impulse model "Salt and Pepper" – to errors at the data transmission stage.

A system was built for the study to store data for analysis. The data acquisition process consists of passing tests lasting 30 seconds with the chosen type of noise and its intensity. Once the data have been accumulated, they can be analyzed to form conclusions.

As a result of the research, the following was obtained: Gaussian noise causes a high complexity of object recognition in the image then the noise "Salt and Pepper"; also with the increase of noise intensity of Gauss linearly increases recognition complexity. This result may be due to the fact that when the image is distorted by noise, the object to be recognized is also exposed to a blurring effect, as all three channels of the image are spoiled. If there is noise "Salt and Pepper" only falling pixels are visible, and the contour of the object will not be distorted.

The results of the research are relevant for use in the medical industry, as there the image is widely used for the diagnosis of patients. Digital images are also used in astronomy for cosmology to study celestial bodies. The results of the study can be

used to assess errors in the processing of digital images, in particular the recognition of objects on them, which emphasizes the relevance of the experiments.

Compliance with Ethical Standards

Funding: This research received no external funding.

Conflict of interest: The author declares no conflict of interest.

Research involving Human Participants and/or Animals: Not applicable.

Informed consent: Not applicable.

REFERENCES

[1] Maier, A.—Syben, C.—Lasser, T.—Riess, C.: A Gentle Introduction to Deep Learning in Medical Image Processing. Zeitschrift für Medizinische Physik, Vol. 29, 2019, No. 2, pp. 86–101, doi: 10.1016/j.zemedi.2018.12.003.

[2] Abdallah, Y. M. Y.—Alqahtani, T.: Research in Medical Imaging Using Image Processing Techniques. In: Zhou, Y. (Ed.): Medical Imaging – Principles and Applications. Intechopen, Rijeka, 2019, doi: 10.5772/intechopen.84360.

[3] He, S.—Ravanbakhsh, S.—Ho, S.: Analysis of Cosmic Microwave Background with Deep Learning. 2018, https://openreview.net/forum?id=B15uoOyvz.

[4] Tkachenko, R.—Tkachenko, P.—Izonin, I.—Tsymbal, Y.: Learning-Based Image Scaling Using Neural-Like Structure of Geometric Transformation Paradigm. In: Hassanien, A. E., Oliva, D. A. (Eds.): Advances in Soft Computing and Machine Learning in Image Processing. Springer, Cham, Studies in Computational Intelligence, Vol. 730, 2018, pp. 537–565, doi: 10.1007/978-3-319-63754-9_25.

[5] Peleshko, D.—Rak, T.—Izonin, I.: Image Superresolution via Divergence Matrix and Automatic Detection of Crossover. International Journal of Intelligent Systems and Applications (IJISA), Vol. 8, 2016, No. 12, pp. 1–8, doi: 10.5815/ijisa.2016.12.01.

[6] Ogawa, K.—Ito, Y.—Nakano, K.: Efficient Canny Edge Detection Using a GPU. 2010 First International Conference on Networking and Computing, 2010, pp. 279–280, doi: 10.1109/IC-NC.2010.13.

[7] Julliand, T.—Nozick, V.—Talbot, H.: Image Noise and Digital Image Forensics. In: Shi, Y. Q., Kim, H. J., Pérez-González, F., Echizen, I. (Eds.): Digital-Forensics and Watermarking. Springer, Cham, Lecture Notes in Computer Science, Vol. 9569, 2016, pp. 3–17, doi: 10.1007/978-3-319-31960-5_1.

[8] Canny, J.: A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-8, 1986, No. 6, pp. 679–698, doi: 10.1109/TPAMI.1986.4767851.

[9] Zhang, C. C.—Fang, J. D.: Edge Detection Based on Improved Sobel Operator. Proceedings of the 2016 International Conference on Computer Engineering and Information Systems, Atlantis Press, 2016, pp. 129–132, doi: 10.2991/ceis-16.2016.25.

[10] Dim, J. R.—Takamura, T.: Alternative Approach for Satellite Cloud Classification: Edge Gradient Application. Advances in Meteorology, Vol. 2013, 2013, Art. No. 584816, doi: 10.1155/2013/584816.

[11] Ansari, M. A.—Kurchaniya, D.—Dixit, M.: A Comprehensive Analysis of Image Edge Detection Techniques. International Journal of Multimedia and Ubiquitous Engineering, Vol. 12, 2017, No. 11, pp. 1–12, doi: 10.14257/ijmue.2017.12.11.01.

[12] Yang, C. Y.—Samani, H.—Ji, N.—Li, C.—Chen, D. B.—Qi, M.: Deep Learning Based Real-Time Facial Mask Detection and Crowd Monitoring. Computing and Informatics, Vol. 40, 2021, No. 6, pp. 1263–1294, doi: 10.31577/cai_2021_6_1263.

[13] Luo, Y.—Duraiswami, R.: Canny Edge Detection on NVIDIA CUDA. 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008, pp. 1–8, doi: 10.1109/CVPRW.2008.4563088.

[14] Mochurad, L.—Kryvinska, N.: Parallelization of Finding the Current Coordinates of the Lidar Based on the Genetic Algorithm and OpenMP Technology. Symmetry, Vol. 13, 2021, No. 4, Art. No. 666, doi: 10.3390/sym13040666.

[15] Kalaiselvi, T.—Sriramakrishnan, P.—Somasundaram, K.: Survey of Using GPU CUDA Programming Model in Medical Image Analysis. Informatics in Medicine Unlocked, Vol. 9, 2017, pp. 133–144, doi: 10.1016/j.imu.2017.08.001.

[16] Pan, L.—Gu, L.—Xu, J.: Implementation of Medical Image Segmentation in CUDA. 2008 International Conference on Information Technology and Applications in Biomedicine, 2008, pp. 82–85, doi: 10.1109/ITAB.2008.4570542.

[17] Smistad, E.—Falch, T. L.—Bozorgi, M.—Elster, A. C.—Lindseth, F.: Medical Image Segmentation on GPUs – A Comprehensive Review. Medical Image Analysis, Vol. 20, 2015, No. 1, pp. 1–18, doi: 10.1016/j.media.2014.10.012.

[18] Gonzalez, R.—Woods, R.: Digital Image Processing. Technosphere, Moscow, 2005 (in Russian).

[19] Liang, H.—Li, N.—Zhao, S.: Salt and Pepper Noise Removal Method Based on a Detail-Aware Filter. Symmetry, Vol. 13, 2021, No. 3, Art. No. 515, doi: 10.3390/sym13030515.

[20] Bansal, M.—Kumar, M.—Kumar, M.: 2D Object Recognition Techniques: State-of-the-Art Work. Archives of Computational Methods in Engineering, Vol. 28, 2021, No. 3, pp. 1147–1161, doi: 10.1007/s11831-020-09409-1.

[21] Lourenço, L. H. A.—Weingaertner, D.—Todt, E.: Efficient Implementation of Canny Edge Detection Filter for ITK Using CUDA. 2012 13th Symposium on Computer Systems, 2012, pp. 33–40, doi: 10.1109/WSCAD-SSC.2012.21.

[22] Mochurad, L.—Boyko, N.: Technologies of Distributed Systems and Parallel Computation: Monograph. Publishing House "Bona", Lviv, Ukraine, 2020.

[23] Lindholm, E.—Nickolls, J.—Oberman, S.—Montrym, J.: NVIDIA Tesla: A Unified Graphics and Computing Architecture. IEEE Micro, Vol. 28, 2008, No. 2, pp. 39–55, doi: 10.1109/MM.2008.31.

[24] Mochurad, L.—Hladun, Y.: Modeling of Psychomotor Reactions of a Person Based on Modification of the Tapping Test. International Journal of Computing, Vol. 20, 2021, No. 2, pp. 190–200, doi: 10.47839/ijc.20.2.2166.

Lesia Mochurad (Ph.D.) is Associate Professor of the Department of Artificial Intelligence, Lviv Polytechnic National University, Lviv, Ukraine.