Politechnika Warszawska

Cyfrowe Przetwarzanie Obrazu 2

DOKUMENTACJA

Projekt 1

Fi1 – Filtr splotowy LoG

Jarosław Affek
gr. 49

Warszawa 2018

1. Tabela z deklaracją zrealizowanych zadań

	Zadanie	Wariant	Punkty
Algorytm	Implementacja podstawowej wersji algorytmu	Bezbłędna	3,75
		Drobne uchybienia	3
		Błędy	2,25
		Poważne błędy	0,75
		Niepoprawna	0
	Parametryzacja rozwiązania	Pełna	2,25
		Częściowa	0,75
		Brak	0
	Implementacja innych realizacji, postaci		1
SUMA			7~7,5/7,5
Implementacja	Podział na metody – zasada pojedynczej odpowiedzialności		1,5
	Informatywne nazwy zmiennych		0,6
	Spójne nazewnictwo, język, styl		0,45
	Komentarze i przypisy		0,45
	Podział na interfejs i warstwę przetwarzania		0,75
	Kod bez powtórzeń		0,75
SUMA			4,5/4,5
Opis teoretyczny	Opis teoretyczny algorytmu		1,5
	Opis implementacji		0,75
	Opis wpływu parametrów na efekt działania		0,75
SUMA			3/3
SUMA CAŁKOWITA			14,5~15/15

2. Wstęp

Celem projektu było zaimplementowanie algorytmu filtracji splotowej Laplacian of Gaussian. Jest to filtr składający się z dwóch filtrów następujących po sobie. Filtr Gaussa jest to filtr rozmywający, jednak w przeciwieństwie do filtru uśredniającego rozkład współczynników w masce takiego filtra nie jest jednostajny tylko gaussowski – oznacza to, że podczas splotu z obrazem uzyskujemy rozmycie „gładkie", bez widocznych linii, ponieważ wartość piksela na wyjściu jest bardziej zależna od pikseli, które są bliżej i coraz mniej od coraz dalszych pikseli w obrębie maski. Natomiast filtr Laplaca jest to filtr, który wykrywa krawędzie. Stosowanie samego filtra Laplaca zazwyczaj nie przynosi pożądanych rezultatów, ponieważ wykryte zostają najdrobniejsze krawędzie, również te pochodzące od szumów, w rezultacie obraz wyjściowy jest bardzo zaszumiony. Dlatego stosuje się Laplacian of Gaussian, obraz po rozmyciu filtrem Gaussa pozbawiony jest drobnych szumów, natomiast większe krawędzie zostają widoczne, zastosowanie następnie filtra Laplaca pozwala wyodrębnić krawędzie w obrazie z pominięciem (przynajmniej częściowym) szumu. Filtr ten ma zastosowanie do automatycznej analizy wizyjnej w różnego rodzaju procesach produkcyjnych, jak również po odpowiednich działaniach arytmetycznych efekt działania tego filtru można wykorzystać do wyostrzania zdjęć.

3. Algorytm

- **Implementacja podstawowej wersji algorytmu**

Zadanie: Należy zaimplementować algorytm filtracji splotowej Laplacian of Gaussian, ze zmiennym rozmiarem jądra (parametr podawany przez użytkownika). Algorytm powinien działać na obrazach w skali szarości lub na poszczególnych kanałach. Należy opisać cel i podać przykład zastosowania tego typu filtracji.

Realizacja:

- Wczytany obraz jest konwertowany z typu uchar (zakres intensywności 0 - 255) na float (zakres intensywności 0 - 1) aby podczas licznych obliczeń nie stracić dokładności, funkcja bierze pod uwagę podany zakres współrzędnych, dzięki czemu konwersja może odbyć się jedynie w ramach tzw. ROI (Region of interest). Ze względu na wymóg niepowtarzania kodu poniższa funkcja została zapisana w bardzo nieoptymalny sposób. Najpierw wykonywana jest pętla i w każdym przejściu pętli sprawdzane są warunki if. Gdyby zastosować na zewnątrz warunek if a następnie w każdym wariancie pętle funkcja byłaby znacznie szybsza i wykonywałaby mniej operacji, wymagałoby to jednak powtórzenia kodu.

```
// convertion from CV_8U -> CV_32F taking into account the region of interest
void rewrite_matrix_to_float(const Mat & source_img, Mat & out, int x1, int y1, int
x2, int y2, int channels_num)
{
	if(channels_num == 3)
		out = Mat(y2 - y1, x2 - x1, CV_32FC3);
	else
		out = Mat(y2 - y1, x2 - x1, CV_32FC1);
	for (int i = x1; i < x2; i++)
		for (int j = y1; j < y2; j++)
```

```
                if (channels_num == 3)
                        for (int k = 0; k < 3; k++)
                                if (source_img.depth() == CV_32F)
                                        out.at<Vec3f>(j - y1, i - x1)[k] =
source_img.at<Vec3f>(j, i)[k];
                                else
                                        out.at<Vec3f>(j - y1, i - x1)[k] = 1.0 / 255 *
source_img.at<Vec3b>(j, i)[k];
                else
                        if (source_img.depth() == CV_32F)
                                out.at<float>(j - y1, i - x1) =
source_img.at<float>(j, i);
                        else
                                out.at<float>(j - y1, i - x1) = 1.0 / 255 *
source_img.at<uchar>(j, i);
}
```

- W podstawowej wersji algorytmu (w następnych punktach przedstawiona będzie dodatkowa wersja) kolejnym krokiem programu jest stworzenie maski Gaussa, cechą maski Gaussa jest to, że suma jej elementów musi być równa 1, aby bilans „energetyczny" obrazu pozostał bez zmian po splocie. Gdyby tak nie było obraz mógłby ulec przyciemnieniu lub rozjaśnieniu. Dlatego w poniższej funkcji po zastosowaniu wzoru Gaussa dla każdego elementu maski konieczne jest wykonanie operacji normalizacji maski. Maska Gaussa może mieć dowolny rozmiar. Elementy w masce wyliczane są wg wzoru:

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

Rysunek 1 [2]

```
// calculate gaussian matrix
void create_gaussian_mask(Mat & out, int sigma_tmp, int kernel_size)
{
        float sigma = (float)sigma_tmp / 10;
        out = Mat(2 * kernel_size + 1, 2 * kernel_size + 1, CV_32FC1);
        float sum = 0, normalize = 0;
        for (int i = -kernel_size; i <= kernel_size; i++)
                for (int j = -kernel_size; j <= kernel_size; j++)
                {
                        out.at<float>(i + kernel_size, j + kernel_size) = 1.0 / (2 *
CV_PI*sigma*sigma)*exp(-((i * i + j * j) / (2 * sigma * sigma)));  // gaussian formula
                        sum = sum + out.at<float>(i + kernel_size, j + kernel_size);
                }
        // normalization - the sum of all elements of the gaussian matrix must be equal
to 1
        normalize = 1.0 / sum;
        for (int i = -kernel_size; i <= kernel_size; i++)
                for (int j = -kernel_size; j <= kernel_size; j++)
                        out.at<float>(i + kernel_size, j + kernel_size) = out.at<float>(i
+ kernel_size, j + kernel_size) * normalize;
}
```

- Kolejnym krokiem jest stworzenie maski Laplaca, z definicji operator Laplaca jest to suma drugich pochodnych z funkcji po x i y. Czyli jest to różnica intensywności między sąsiednimi pikselami. Dlatego maska Laplaca jest to maska 3x3 – dwie najczęściej stosowane maski są przedstawione poniżej.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

Rysunek 2 [1]

W projekcie stosowana jest maska znajdująca się po prawej stronie rysunku 2 (po splocie obrazu z taką masą krawędzie są intensywniejsze niż po splocie maską po lewej stronie) oraz jej modyfikacje (rozmiar podawany jako parametr). Większa maska Laplaca tworzona jest analogicznie do tej 3x3 – wszędzie wstawiane są -1 (lub 1) a w centralnej części suma wszystkich jedynek (z odwrotnym znakiem). Suma elementów w masce Laplaca wynosi 0.[1]

```
// calculate laplace matrix - the sum of all elements of the gaussian matrix must be
equal to 0 - evereywhere 1 except the central element
void laplace(Mat & out, int kernel_size)
{
	out = Mat(2 * kernel_size + 1, 2 * kernel_size + 1, CV_32FC1);
	for (int i = -kernel_size; i <= kernel_size; i++)
		for (int j = -kernel_size; j <= kernel_size; j++)
			out.at<float>(i + kernel_size, j + kernel_size) = 1;
	out.at<float>(kernel_size, kernel_size) = -((kernel_size * 2 + 1)*(kernel_size *
2 + 1) - 1);
}
```

Algorytm składa się z następujących kroków:
- wykonanie splotu obrazu wejściowego z maską Gaussa
- wykonanie splotu obrazu wyjściowego z poprzedniego punktu z maską Laplaca

Wykorzystana do tego jest funkcja splotu, więcej o splocie w części „Opis teoretyczny”. Funkcja splotu jest dość rozbudowana ze względu na konieczność obsługi dwóch przypadków obrazów – jednokanałowych i wielokanałowych, ze względu na charakterystykę OpenCV, aby edytować i odczytywać wartości pikseli w macierzy Mat trzeba podać jej typ (float lub Vec3f), jako że nie da się tych typów danych podać jako argumentów funkcji konieczne było stworzenie dużej instrukcji warunkowej if dla obydwu wersji obrazów. Aby wykonać operację splotu należy stworzyć 4 pętle (2 do przechodzenia w 2 wymiarach po obrazie i wewnątrz 2 przechodzące po masce). Następnie wewnątrz pętli następuje sumowanie iloczynów elementów maski i obrazu. Ostatecznie suma ta jest wpisywana do piksela.

```
// convolution of source image and mask
void convolution_img(const Mat & source_img, Mat & out, Mat mask)
{
	int kernel_size = (mask.rows - 1) / 2;
	if (source_img.channels() == 1)
	{
		out = Mat(source_img.rows - 2 * kernel_size, source_img.cols - 2 *
kernel_size, CV_32FC1);
		for (int i = kernel_size; i < source_img.rows - kernel_size; i++)
			for (int j = kernel_size; j < source_img.cols - kernel_size; j++)
// navigating the source image
```

```
                        {
                                float sum = 0;
                                for (int k = -kernel_size; k <= kernel_size; k++)
                                        for (int m = -kernel_size; m <= kernel_size; m++)
// navigating the kernel
                                        {
                                                float component = 0;
                                                component = source_img.at<float>(i + k, j + m)
* mask.at<float>(-k + kernel_size, -m + kernel_size);
                                                sum += component;
                                        }
                                out.at<float>(i - kernel_size, j - kernel_size) = sum;
                        }
        }
        else
        {
                out = Mat(source_img.rows - 2 * kernel_size, source_img.cols - 2 *
kernel_size, CV_32FC3);
                for (int i = kernel_size; i < source_img.rows - kernel_size; i++)
                        for (int j = kernel_size; j < source_img.cols - kernel_size; j++)
// navigating the source image
                        {
                                float sum[3] = { 0,0,0 };
                                for (int k = -kernel_size; k <= kernel_size; k++)
                                        for (int m = -kernel_size; m <= kernel_size; m++)
// navigating the kernel
                                        {
                                                float component[3] = { 0,0,0 };
                                                for (int n = 0; n < 3; n++)
                                                {
                                                        component[n] = source_img.at<Vec3f>(i +
k, j + m)[n] * mask.at<float>(-k + kernel_size, -m + kernel_size);
                                                        sum[n] += component[n];
                                                }
                                        }
                                if(R_channel == 1)
                                        out.at<Vec3f>(i - kernel_size, j - kernel_size)[2] =
sum[2];
                                if (G_channel == 1)
                                        out.at<Vec3f>(i - kernel_size, j - kernel_size)[1] =
sum[1];
                                if (B_channel == 1)
                                        out.at<Vec3f>(i - kernel_size, j - kernel_size)[0] =
sum[0];
                        }
        }
}
```

Wynik działania programu w postaci obrazów po zastosowaniu filtrów przedstawiony zostanie w punkcie „parametryzacja rozwiązania” gdyż będzie można przedstawić różnice w działaniu programu w zależności od różnych parametrów.

- **Parametryzacja rozwiązania**

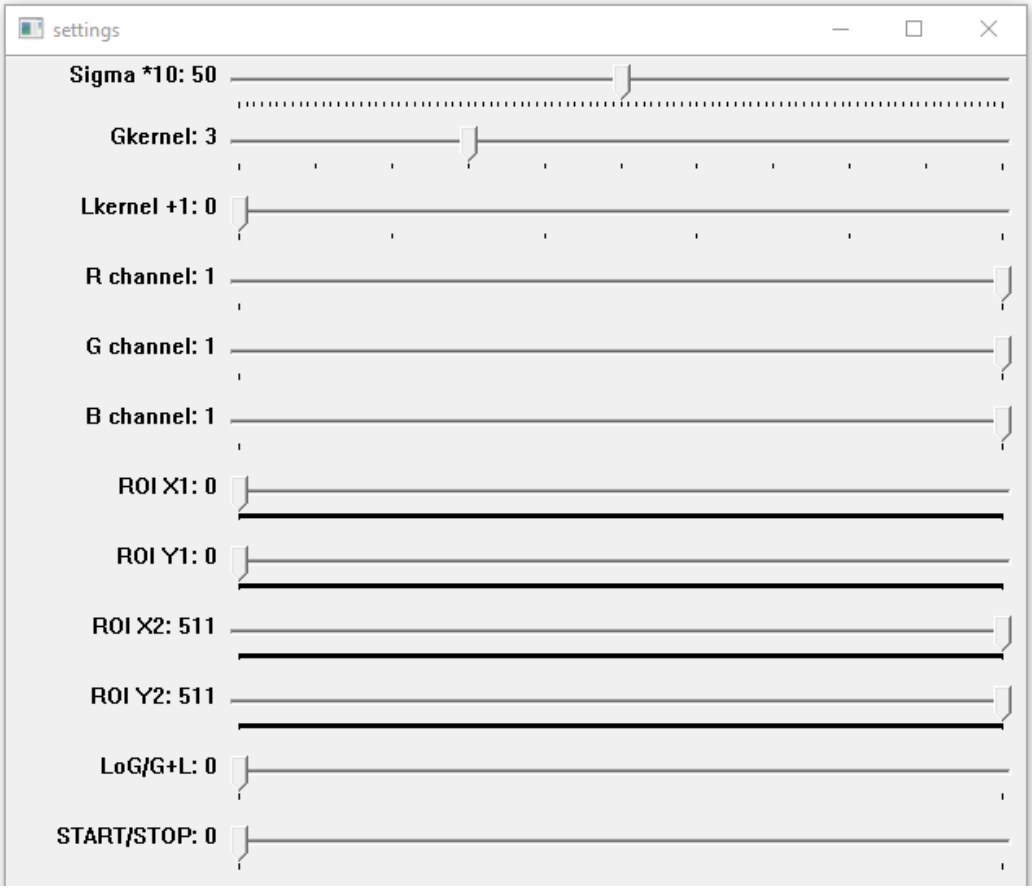
Dzięki zastosowaniu GUI biblioteki OpenCV cała interakcja użytkownika z programem opiera się na 12 suwakach widocznych po otwarciu programu. W programie parametrami możliwymi do regulacji przez użytkownik są:

- parametr sigma (występujący w wzorze Gaussa: od 0 do 10)
- rozmiar jądra maski Gaussa (0 - 10)
- rozmiar jądra maski Laplaca (1 - 6)
- włączenie przetwarzania obrazu dla kanału R
- włączenie przetwarzania obrazu dla kanału G
- włączenie przetwarzania obrazu dla kanału B
- wybór parametru X1 ROI (wybór ROI poprzez ustawienie 2 punktów (X1,Y1) i (X2,Y2))
- wybór parametru Y1 ROI
- wybór parametru X2 ROI
- wybór parametru Y2 ROI
- wybór algorytmu przetwarzania (kolejny będzie opisany w następnym punkcie)
- START/STOP włączenie/wyłączenie przetwarzania

Dzięki zastosowaniu suwaków wybór parametrów jest łatwiejszy i znacznie szybszy oraz eliminuje się możliwość wprowadzenia niepoprawnych danych przez użytkownika. Obszar ROI należy wprowadzić tak, aby X1<X2 oraz Y1<Y2 – w przeciwnym wypadku program przeliczy obraz dla poprzednio ustawionych poprawnych ustawień. Jako obraz źródłowy dla wszystkich operacji będzie zastosowany obraz widoczny na rysunku poniżej.

Rysunek 3

Panel opisanych wyżej ustawień widoczny jest na rys. 4.



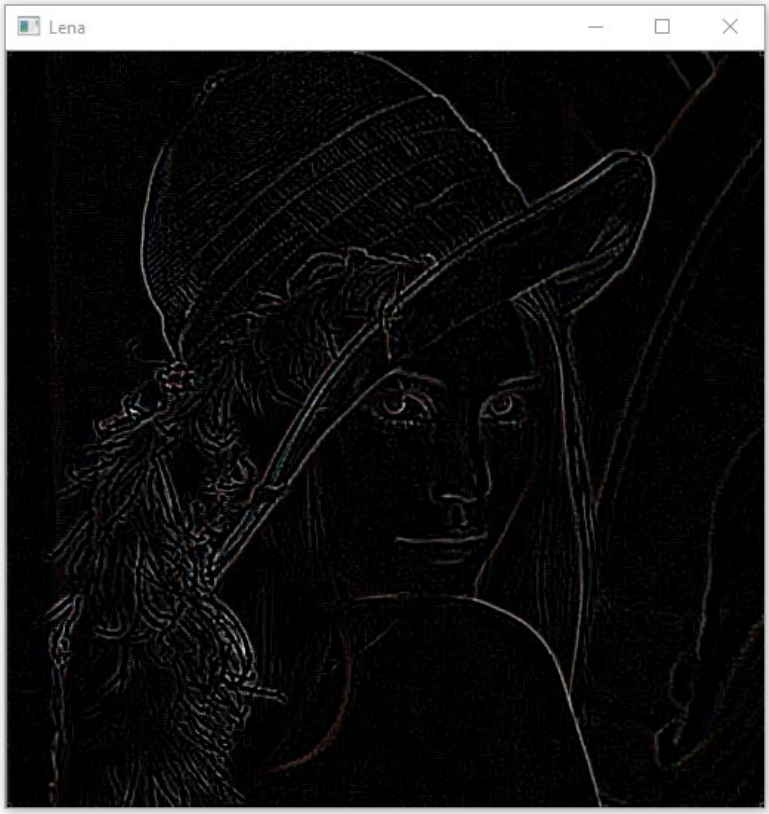
Rysunek 4

Należy wspomnieć, że ustawianie rozmiaru jądra maski jest wykonywane w taki sposób, że wartość ustawiona na suwaku jest to odległość od środka maski do jej krawędzi – eliminuje to możliwe błędne ustawienie maski o parzystym boku przez użytkownika.

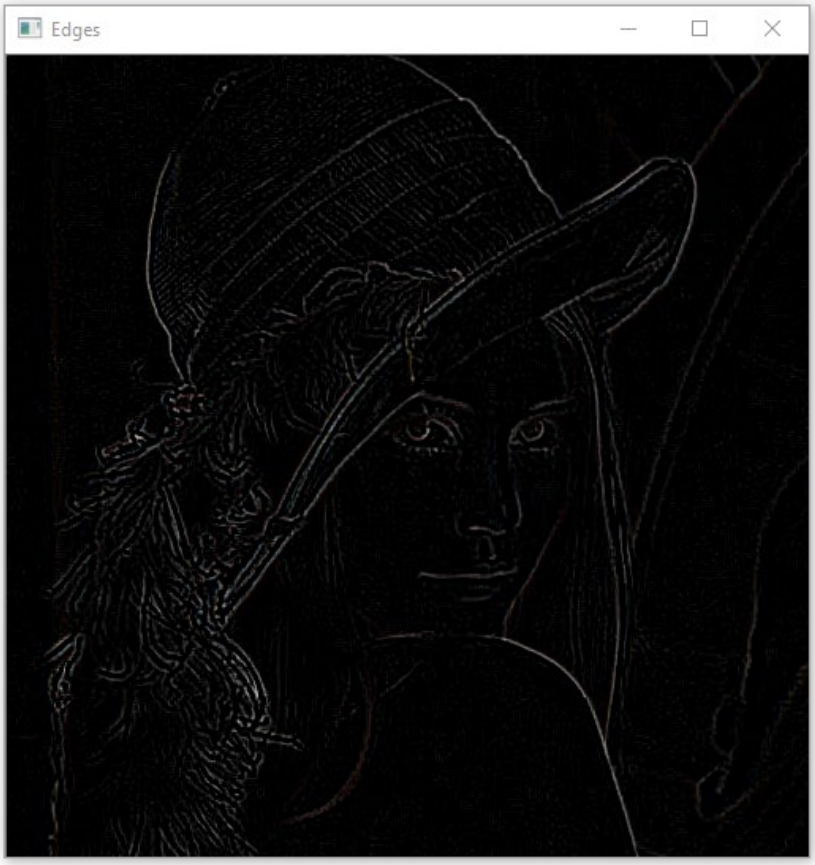
Najpierw jako **dowód poprawności** działania algorytmu przedstawiony zostanie efekt działania wbudowanych funkcji w OpenCV: GaussianBlur + Laplacian.

```
int main()
{
	namedWindow("Lena", WINDOW_AUTOSIZE);
	Mat lena = imread("lena1.jpg");
	GaussianBlur(lena, lena, Size(3, 3), 1.5, 1.5);
	Laplacian(lena, lena, CV_8U, 3,1);
	imshow("Lena",lena);
	waitKey(0);
	return 0;
}
```

Efekt działania powyższej funkcji przedstawia rys. 5. Dla porównania poniżej na rys. 6 widać efekt działania zaimplementowanego algorytmu.

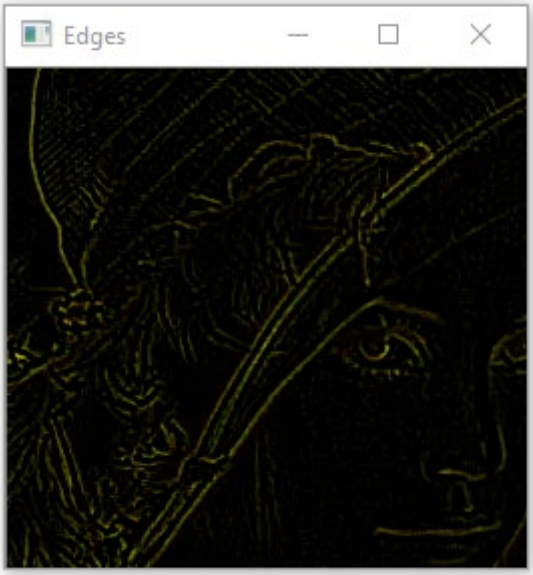


Rysunek 5



Rysunek 6

Jak widać różnice są bardzo niewielkie i wynikają prawdopodobnie z normalizacji maski Gaussa. Obydwa efekty są uzyskane dla wszystkich 3 kanałów RGB, wartości sigma = 1.5 oraz rozmiarach jąder obu masek 3x3. Dzięki możliwości edycji parametrów możliwe jest działanie tylko na wybranym kanale obrazu (lub wybranych dwóch lub trzech) oraz jedynie w obszarze ROI. Przykład rys. 7 dla kanałów R oraz G i ograniczonym ROI.



Rysunek 7

Dokładny opis różnic w efektach związany ze zmianą parametrów znajduje się w części „Opis teoretyczny".

- **Implementacja innych realizacji, postaci**

Jako drugi algorytm zastosowany został algorytm, który polega na wyznaczeniu jednej maski Laplacian of Gaussian. W tej metodzie dzięki zastosowaniu wzoru Laplaca (suma drugich pochodnych) na wzorze Gaussa (więcej w części „Opis teoretyczny") otrzymano jeden wzór, z którego można obliczyć maskę filtra, a następnie wykonać tylko jeden splot z obrazem.

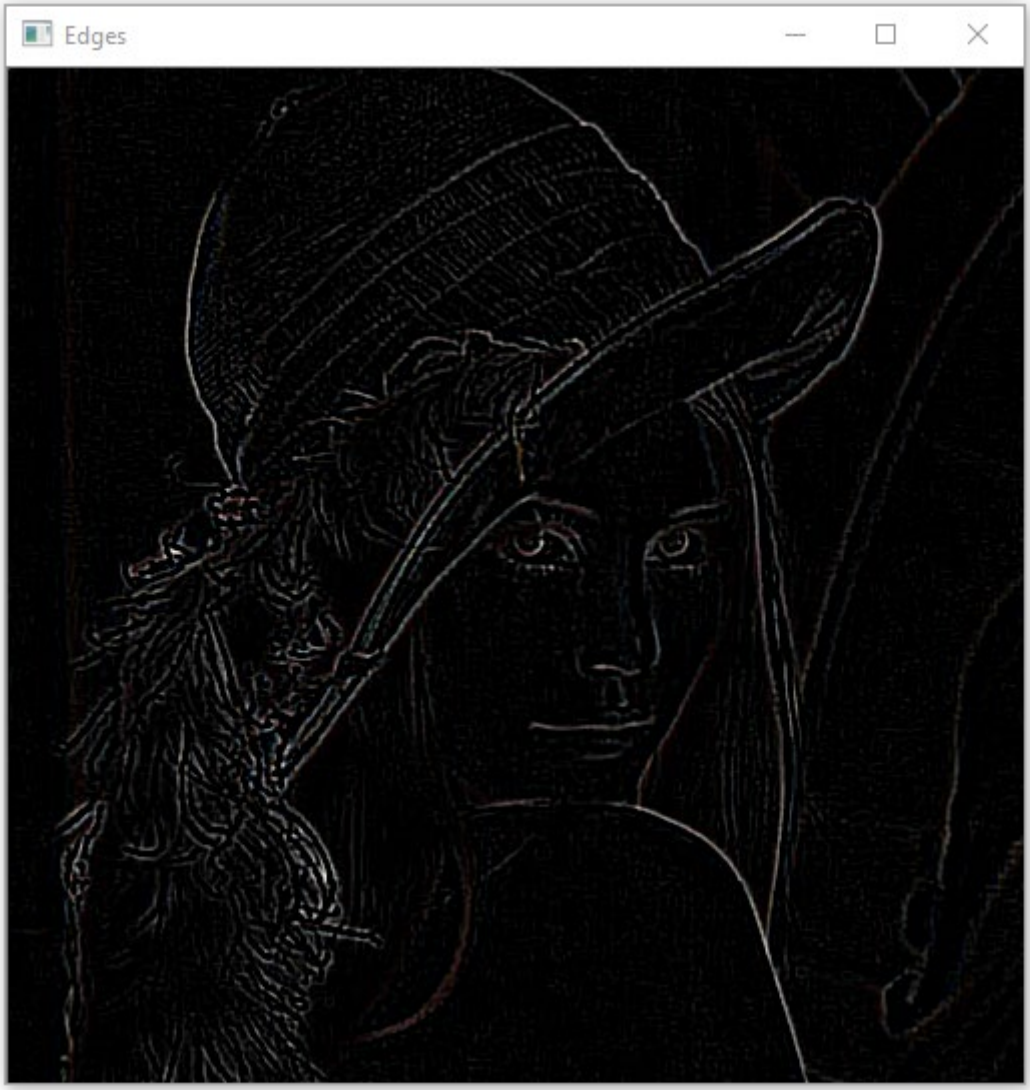
$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Rysunek 8

Metoda ta ma jednak wadę – istnieje możliwość edycji tylko jednego wymiaru maski (ponieważ istnieje tylko jedna maska), dlatego wpływ na efekt końcowy jest mniejszy niż przy poprzedniej metodzie. Kod tej funkcji przedstawiony jest poniżej – jest on stosunkowo prosty – wykorzystuje jeden wzór na LoG, a większa część funkcji jest to normalizacja powstałej maski, aby suma elementów wynosiła 0 oraz ich wartości były wystarczająco duże (największy element aby miał wartość 1/-1).

```
// another implementation - calculation of tha Laplacian of Gaussian mask using known
LoG formula
void laplacian_of_gaussian(Mat & out, int sigma_tmp, int kernel_size)
{
	float sigma = (float)sigma_tmp / 10;
	float max_element = 0;
	out = Mat(2 * kernel_size + 1, 2 * kernel_size + 1, CV_32FC1);
	float sum = 0, normalize_mask = 0;
	for (int i = -kernel_size; i <= kernel_size; i++)
		for (int j = -kernel_size; j <= kernel_size; j++)
		{
			out.at<float>(i + kernel_size, j + kernel_size) = -1.0 /
(CV_PI*pow(sigma,4))*(float)(1-(float)(i*i+j*j)/(2*sigma*sigma))*exp(-((float)(i * i +
j * j) / (2 * sigma * sigma)));  // laplacian of gaussian formula
			sum += out.at<float>(i + kernel_size, j + kernel_size);
		}
	//  normalize - sum have to be 0
	normalize_mask = -sum / (pow((2 * kernel_size + 1), 2));
	for (int i = -kernel_size; i <= kernel_size; i++)
		for (int j = -kernel_size; j <= kernel_size; j++)
		{
			out.at<float>(i + kernel_size, j + kernel_size) = out.at<float>(i
+ kernel_size, j + kernel_size) + normalize_mask;
			if (abs(out.at<float>(i + kernel_size, j + kernel_size)) >
max_element)
				max_element = abs(out.at<float>(i + kernel_size, j +
kernel_size));
		}
	sum = 0;
	for (int i = -kernel_size; i <= kernel_size; i++)
		for (int j = -kernel_size; j <= kernel_size; j++)
		{
			out.at<float>(i + kernel_size, j + kernel_size) = out.at<float>(i
+ kernel_size, j + kernel_size) / max_element;
			sum += out.at<float>(i + kernel_size, j + kernel_size);
		}
}
```

Algorytm ten uruchamiamy zmieniając położenie suwaka G+L/LoG. Przy odpowiednim doborze parametrów (sigma 1.3, jądro 5x5, RGB) możemy uzyskać efekt bardzo zbliżony do efektów na rys. 4 i 5. Efekt działania algorytmu przedstawia rys. 9.



Rysunek 9

4. Implementacja

- **Podział na metody – zasada pojedynczej odpowiedzialności**

Każda funkcja napisana w programie ma konkretne zadanie i wywołanie jej ma zawsze na celu wykonanie jednej i tej samej czynności. Są to np. funkcja tworząca maskę Laplaca, wykonująca operację splotu, funkcja wyświetlająca okna i suwaki, jak również funkcje, które mają za zadanie jedynie wywołać w odpowiedniej kolejności inne funkcje (funkcje główne oraz funkcja realizująca określone działanie po zmianie położenia suwaka). Przykład – funkcja wyświetlająca:

```
// create trackbars to edit settings of the Laplacian of Gaussian transformation
void trackbars(int img_rows, int img_cols, Mat source_img, Mat trackbars_img)
{
	namedWindow("settings", CV_WINDOW_NORMAL);
	namedWindow("Original", CV_WINDOW_NORMAL);
	createTrackbar("Sigma *10", "settings", &gaussian_sigma, 100);
	createTrackbar("Gkernel", "settings", &gaussian_kernel, 10);
```

```
	createTrackbar("Lkernel +1", "settings", &laplace_kernel, 5);
	createTrackbar("R channel", "settings", &R_channel, 1);
	createTrackbar("G channel", "settings", &G_channel, 1);
	createTrackbar("B channel", "settings", &B_channel, 1);
	createTrackbar("ROI X1", "settings", &roi_x1, img_cols - 2);
	createTrackbar("ROI Y1", "settings", &roi_y1, img_rows - 2);
	createTrackbar("ROI X2", "settings", &roi_x2, img_cols - 1);
	createTrackbar("ROI Y2", "settings", &roi_y2, img_rows - 1);
	createTrackbar("G+L/LoG", "settings", &choose_alg, 1);
	createTrackbar("START/STOP", "settings", &start_calculation, 1, on_trackbar);
	imshow("settings", trackbars_img);
	imshow("Original", source_img);
}
```

Funkcja uruchamiana zmianą położenia suwaka START/STOP - zależnie od wybranego wariantu wywołuje odpowiedni algorytm:

```
// function called when user move the START/STOP trackbar
void on_trackbar(int, void*)
{
	if (start_calculation == 1)
	{
		if (roi_x1 < roi_x2 && roi_y1 < roi_y2)
			rewrite_matrix_to_float(float_img, ROI_img, roi_x1, roi_y1,
roi_x2, roi_y2,float_img.channels());
		if (choose_alg == 1)
		{
			laplacian_of_gaussian(laplace_mask, gaussian_sigma, laplace_kernel
+ 1);    // use with laplacian_of_gaussian method
			convolution_img(ROI_img, log_img, laplace_mask);
					// use with laplacian_of_gaussian method
		}
		else
		{
			create_gaussian_mask(gauss_mask, gaussian_sigma, gaussian_kernel);
// gaussian + laplacian method
			laplace(laplace_mask, laplace_kernel+1);
			// gaussian + laplacian method
			convolution_img(ROI_img, gaussian_img, gauss_mask);
		    // gaussian + laplacian method
			convolution_img(gaussian_img, log_img, laplace_mask);
		// gaussian + laplacian method
		}
		display(1, log_img);
	}
	else
		display(0, log_img);
}
```

- **Informatywne nazwy zmiennych**

Każda zmienna ma jednoznacznie kojarzącą się nazwę i nazwa ta sugeruje przeznaczenie danej zmiennej. Również funkcje mają nazwy takie aby możliwie łatwo było się dowiedzieć co dana funkcja robi, nie analizując dogłębnie jej treści. Przykłady:

```
int laplace_kernel, gaussian_kernel, roi_x1, roi_x2, roi_y1, roi_y2,
start_calculation, gaussian_sigma, R_channel, G_channel, B_channel, choose_alg;
```

```
// defining initial values
void define_variables();
// calculate gaussian matrix
void create_gaussian_mask(Mat & out, int sigma_tmp, int kernel_size);
```

- **Spójne nazewnictwo, język, styl**

Język i styl są spójne, funkcję mają nazwy tworzone wg tych samych reguł (małe litery, słowa oddzielone podkreślnikami), cały kod wraz z komentarzami jest w języku angielskim, formatowanie pętli, funkcji, ustawienia klamer i wcięcia są takie same za każdym razem. Przykład:

```
// defining initial values
void define_variables(const Mat & source_img)
{
	laplace_kernel = 0;
	gaussian_kernel = 1;
	roi_x1 = 0;
	roi_y1 = 0;
	roi_x2 = source_img.cols - 1;
	roi_y2 = source_img.rows - 1;
	start_calculation = 0;
	gaussian_sigma = 50;
	R_channel = 1;
	G_channel = 1;
	B_channel = 1;
	choose_alg = 0;
}
```

- **Komentarze i przypisy**

W kodzie oprócz nazwy funkcji informującej o jej działaniu znajdują się również komentarze, które jasno opisują każdą funkcję, trudniejsze zmienne oraz wszystkie wątpliwe momenty oraz zmienne globalne są wyjaśnione w komentarzach. Ponadto komentarze pełnią funkcję porządkowania kodu (oddzielają od siebie poszczególne bloki). Przykład:

```
extern int laplace_kernel, gaussian_kernel, roi_x1, roi_x2, roi_y1, roi_y2, key,
start_calculation, gaussian_sigma, R_channel, G_channel, B_channel, choose_alg;
//laplace_kernel           define kernel size of the laplacian mask: length of the
                           side of the matrix = 2 * laplace_kernel + 1
//gaussian_kernel          define kernel size of the gaussian mask: length of the side
                           of the matrix = 2 * gaussian_kernel + 1
//roi_x1, roi_y1, roi_x2, roi_y2        coordinates of the left and right bottom
                                        corner of the region of interest
//start_calculation        starts and stops the program
//gaussian_sigma           sigma coefficient in the gaussian formula
//R_channel                define on which channel use algorythm
//G_channel                define on which channel use algorythm
//B_channel                define on which channel use algorythm
//choose_alg               define which algorythm (LoG or Gauss + Laplace) will start

extern Mat gauss_mask, gaussian_img, log_img, laplace_mask, ROI_img, float_img,
loaded_img;
//gauss_mask               gaussian matrix
//laplace_mask             laplacian matrix
//loaded_img               the original photo loaded from the file
//float_img                loaded_img after convertion from uchar to float
```

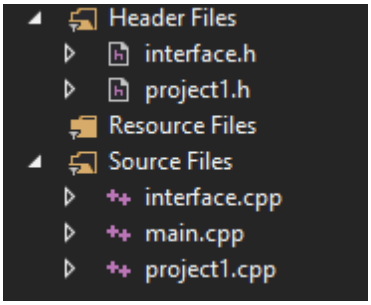
```
//ROI_img                  region of interest of the float image
//log_img                  image after laplacian of gaussian operation

//declared as global variables due to the gui environment requirements in opencv
(trackbars)

//*************************************************************************************
// defining initial values
void define_variables();
//-------------------------------------------------------------------------------------
// calculate gaussian matrix
void create_gaussian_mask(Mat & out, int sigma_tmp, int kernel_size);
//-------------------------------------------------------------------------------------
// convertion from CV_8U -> CV_32F taking into account the region of interest
void rewrite_matrix_to_float(const Mat & source_img, Mat & out, int x1, int y1, int
x2, int y2);
```

- **Podział na interfejs i warstwę przetwarzania**

Cały projekt podzielony jest na 5 plików: main.cpp, project1.cpp, project1.h, interface.cpp, interface.h. W pliku main jest jedynie wywołanie funkcji z pliku project1, natomiast wszystkie funkcje związane z wyświetlaniem i wczytywaniem danych (z suwaków) znajdują się w osobnym pliku i są podzielone wg wykonywanych zadań. W plikach project1 znajduje się warstwa przetwarzania, a w interface – interfejs.



Rysunek 10

- **Kod bez powtórzeń**

Kod jest napisany w taki sposób aby nie było konieczne tworzenie dwa razy tej samej sekwencji wierszy robiących to samo. Jak zostało wspomniane w części „algorytm" nie zawsze jest to uzasadnione, ponieważ czasami powtórzenie kilku wierszy spowodowałoby poprawienie wydajności programu (wykonanie wielokrotnie mniej operacji), jednak byłoby to niezgodne z wymaganiami odnośnie projektu. Jeżeli jakiś element kodu jest wykonywany kilka razy (np. `convolution_img`) to umieszczony został w oddzielnej funkcji i wywoływany jest w pętli.

5. Opis teoretyczny

- **Opis teoretyczny algorytmu**

Jak zaznaczono we wstępie oraz w części „algorytm” operacja Laplacian of Gaussian może być wykonana na dwa zaprezentowane sposoby. W pierwszym sposobie tworzona jest maska filtru Gaussa zgodnie ze wzorem:

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

Rysunek 11 [2]

Przykładowa maska 7x7 uzyskana w ten sposób znormalizowana aby suma elementów równa była 1 przedstawiona jest na rys. 12.

0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00038771	0.01330373	0.11098164	**0.22508352**	0.11098164	0.01330373	0.00038771
0.00019117	0.00655965	0.05472157	0.11098164	0.05472157	0.00655965	**0.00019117**
0.00002292	0.00078633	0.00655965	0.01330373	0.00655965	0.00078633	0.00002292
0.00000067	0.00002292	**0.00019117**	0.00038771	**0.00019117**	0.00002292	0.00000067

Rysunek 12 [3]

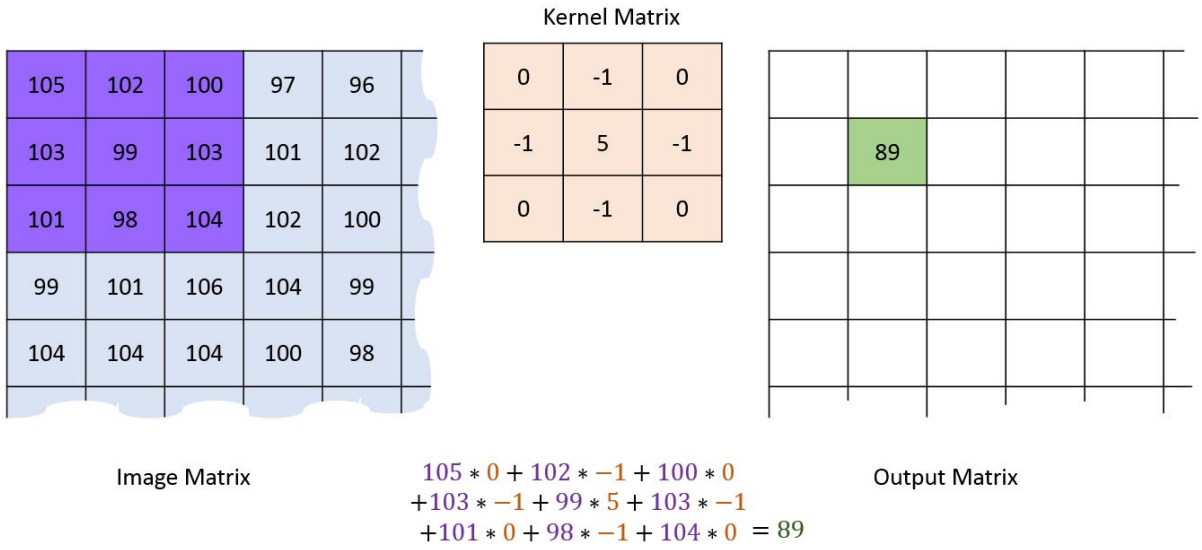
Kolejnym krokiem jest stworzenie maski Laplaca, sposób jej tworzenia oraz przykład maski 3x3 przedstawiony był w części „algorytm”. Na poniższym rysunku przedstawiono rozszerzoną maskę Laplaca do rozmiaru 5x5.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & -24 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Rysunek 13

Aby uzyskać wyjściowy obraz, wykonujemy operację splotu obrazu wejściowego z maską Gaussa, następnie uzyskany obraz ponownie za pomocą splotu z maską tym razem Laplaca przekształcamy na ostateczny efekt.

Splot obrazu z maską jest wykonywany zgodnie z algorytmem przedstawionym na ilustracji poniżej:



Rysunek 14 [5]

Jak widać jest to dość prosta operacja, jednak wymagająca w kodzie aż 4 zagnieżdżonych pętli (dwie to przechodzenia po obrazie i wewnątrz dwie do przechodzenia po masce). Ponadto można zauważyć, że operacja ta nie definiuje wartości pikseli brzegowych, dlatego w zastosowanym algorytmie obraz wyjściowy jest zawsze mniejszy z każdej strony o podawany rozmiar maski (czyli odległość centralnego piksela od brzegu maski).

W drugim przedstawionym algorytmie (opisany również w części „algorytm") tworzona jest jedna maska zgodnie ze wzorem:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$

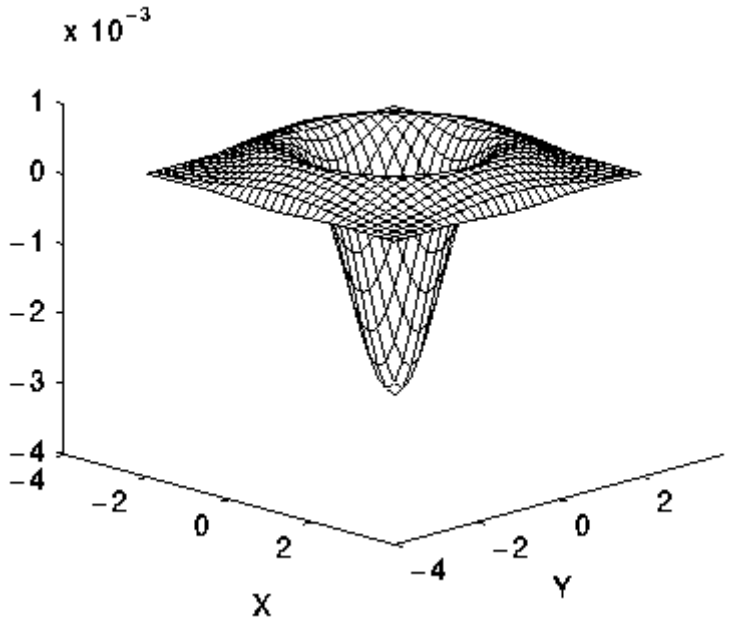
Rysunek 15

Wzór ten wynika z zastosowania sumy drugich pochodnych po x i po y (operator Laplaca) z wzoru Gaussa. Przykładowa maska obliczona w ten sposób, znormalizowana do wartości całkowitych przedstawiona jest na rys. 16

0	1	1	2	2	2	1	1	0
1	2	4	5	5	5	4	2	1
1	4	5	3	0	3	5	4	1
2	5	3	-12	-24	-12	3	5	2
2	5	0	-24	-40	-24	0	5	2
2	5	3	-12	-24	-12	3	5	2
1	4	5	3	0	3	5	4	1
1	2	4	5	5	5	4	2	1
0	1	1	2	2	2	1	1	0

Rysunek 16 [1]

Jak widać maska posiada zarówno elementy dodatnie jak również ujemne a ich rozłożenie w przestrzeni 3D przypominałoby kapelusz – rys. 17.



Rysunek 17 [1]

Stąd nazwa funkcji, której Laplacian of Gaussian jest uogólnieniem – „Mexican hat wavelet” [4].

Patrząc na wykres tej funkcji widać dlaczego działa tak jak działa – czyli wykrywa krawędzie. W momencie gdy badany piksel okazuje się być na krawędzi, czyli ma wartość znacząco inną niż otoczenie to po zastosowaniu takiej maski różnice między tym pikselem a otoczeniem jeszcze bardziej się zwiększają (ze względu na duże różnice w wartości elementów maski, a przede wszystkim przeciwne znaki), natomiast gdy maska obejmuje kawałek obrazu o zerowym gradiencie to jako, że suma elementów maski równa się zero to suma iloczynów elementów maski oraz identycznych pikseli na obrazie będzie wynosić również 0 – czyli obraz wyjściowy będzie czarny dla obszarów pozbawionych gradientu (krawędzi).

- **Opis implementacji**

Kod wraz z opisem implementacji znajduje się w części „algorytm”. W programie wykorzystano wzory zamieszczone powyżej. Wzór Gaussa przy tworzeniu maski Gaussa znajduje się w funkcji `create_gaussian_mask`:

```
out.at<float>(i + kernel_size, j + kernel_size) = 1.0 / (2 * CV_PI*sigma*sigma)*exp(-
((i * i + j * j) / (2 * sigma * sigma)));
```

Wyznaczanie maski Laplaca zostało już przedstawione wcześniej i polega tylko na wpisaniu 1/-1 w każdy element macierzy oprócz centralnego, który jest sumą tych jedynek z przeciwnym znakiem:

```
// calculate laplace matrix - the sum of all elements of the gaussian matrix must be
equal to 0 - evereywhere 1 except the central element
void laplace(Mat & out, int kernel_size)
{
	out = Mat(2 * kernel_size + 1, 2 * kernel_size + 1, CV_32FC1);
	for (int i = -kernel_size; i <= kernel_size; i++)
		for (int j = -kernel_size; j <= kernel_size; j++)
			out.at<float>(i + kernel_size, j + kernel_size) = 1;
	out.at<float>(kernel_size, kernel_size) = -((kernel_size * 2 + 1)*(kernel_size *
2 + 1) - 1);
}
```

Zastosowanie wzoru na formułę Laplacian of Gaussian znajduje się w funkcji `laplacian_of_gaussian`:

```
out.at<float>(i + kernel_size, j + kernel_size) = -1.0 /
(CV_PI*pow(sigma,4))*(float)(1-(float)(i*i+j*j)/(2*sigma*sigma))*exp(-((float)(i * i +
j * j) / (2 * sigma * sigma)));
```

Funkcja splotu została zacytowana i opisana w części „algorytm”.

Dzięki zastosowaniu suwaków, możliwe stało się wygodne i przede wszystkim szybkie zmienianie parametrów i podgląd efektów, bez konieczności wpisywania w konsolę tekstu i uruchamiania programu od nowa. Funkcja on_trackbar jest automatycznie wywoływana natychmiast po zmianie położenia suwaka START/STOP.

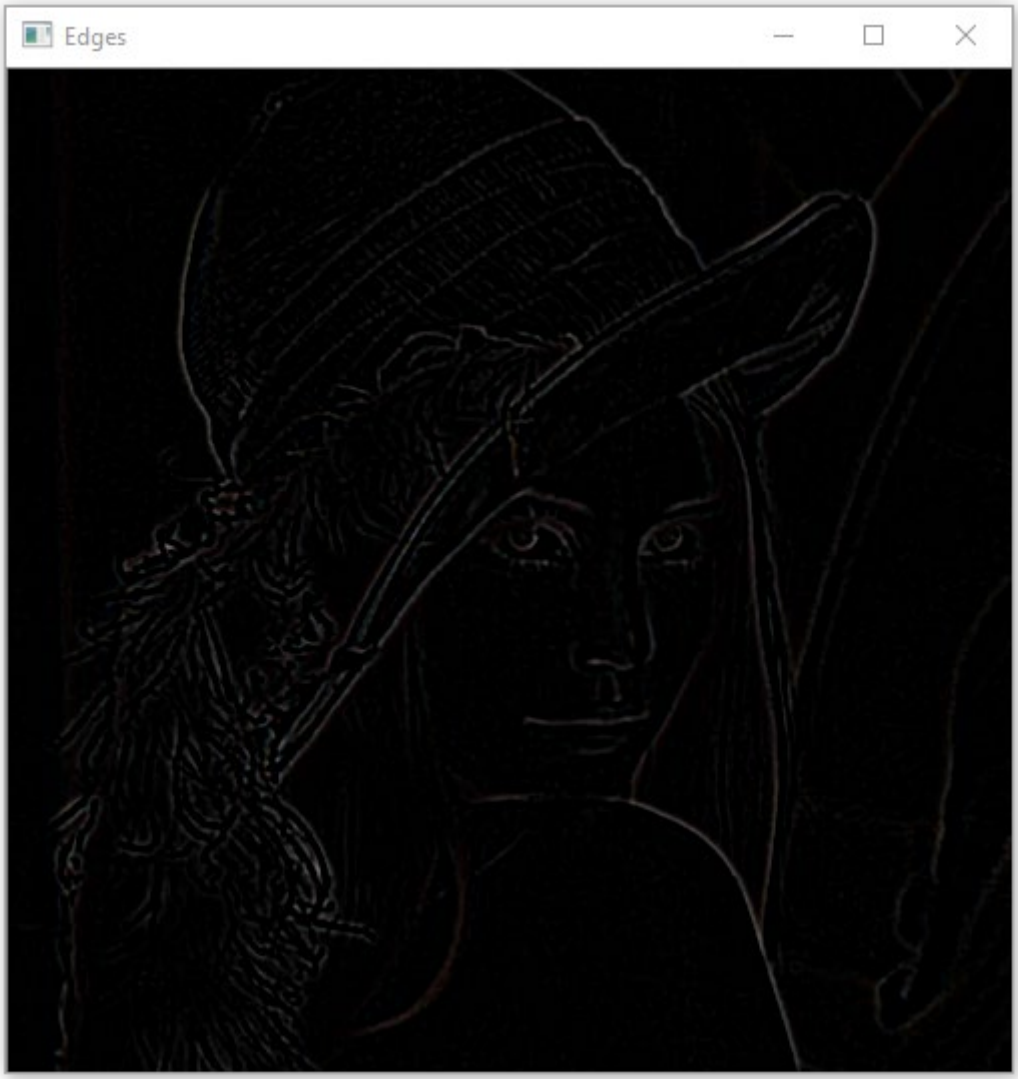
```
// function called when user move the START/STOP trackbar
void on_trackbar(int, void*)
{
	if (start_calculation == 1)
	{
		if (roi_x1 < roi_x2 && roi_y1 < roi_y2)
			rewrite_matrix_to_float(float_img, ROI_img, roi_x1, roi_y1,
roi_x2, roi_y2,float_img.channels());
		if (choose_alg == 1)
		{
			laplacian_of_gaussian(laplace_mask, gaussian_sigma, laplace_kernel
+ 1);    // use with laplacian_of_gaussian method
			convolution_img(ROI_img, log_img, laplace_mask);
				// use with laplacian_of_gaussian method
		}
		else
		{
			create_gaussian_mask(gauss_mask, gaussian_sigma, gaussian_kernel);
// gaussian + laplacian method
			laplace(laplace_mask, laplace_kernel+1);
			// gaussian + laplacian method
			convolution_img(ROI_img, gaussian_img, gauss_mask);
		    // gaussian + laplacian method
			convolution_img(gaussian_img, log_img, laplace_mask);
		// gaussian + laplacian method
		}
		display(1, log_img);
	}
	else
		display(0, log_img);
}
```

Niestety GUI w bibliotece OpenCV jest bardzo ograniczone i nie występują tam przyciski. Również konieczność deklaracji globalnych zmiennych jest niepożądana, ale konieczna, ze względu na zasadę działania suwaków, które edytują bezpośrednio wartość zmiennej, która musi być znana w całej przestrzeni programu.

- **Opis wpływu parametrów na efekt działania**

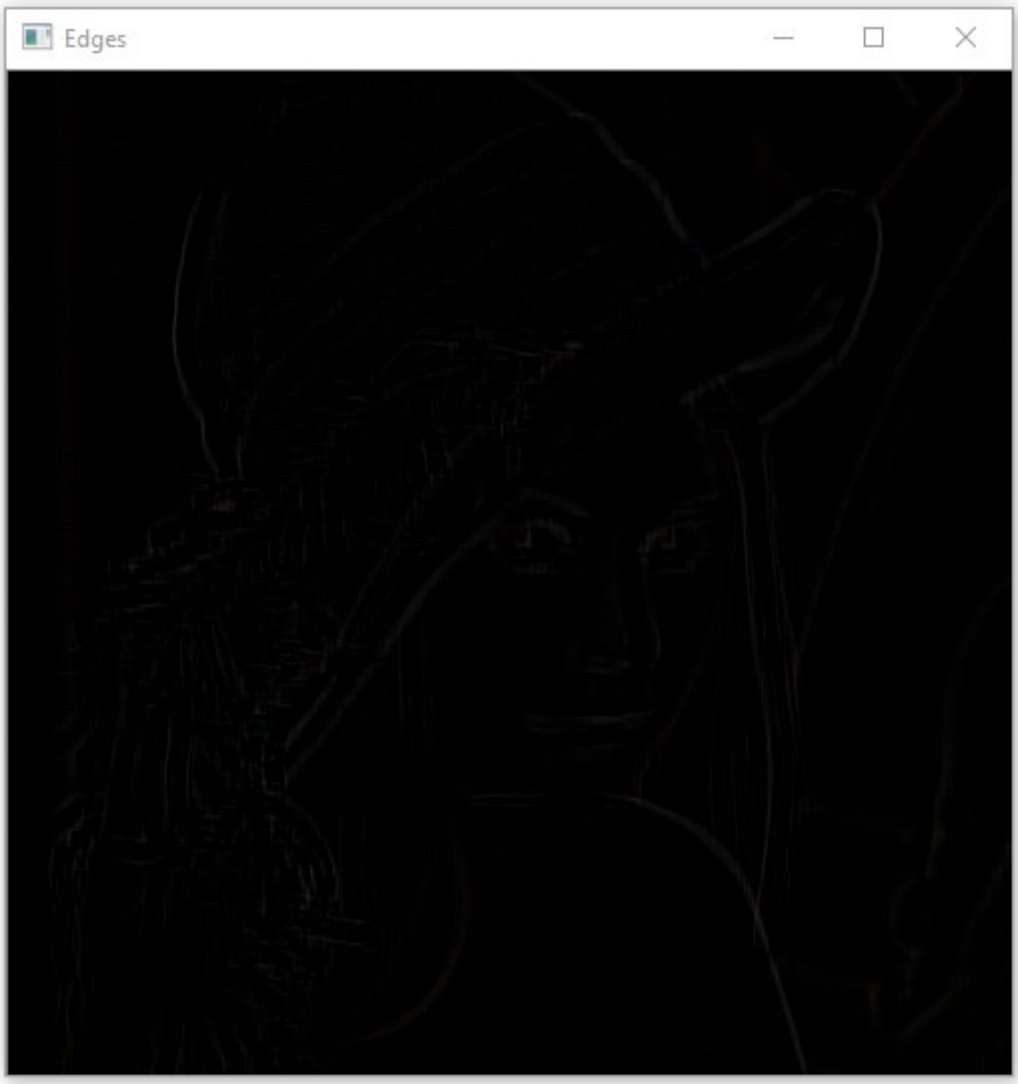
Liczba i opis parametrów możliwych do edycji była przedstawiona w części „algorytm". Tutaj zostaną jedynie przedstawione efekty zmian oraz ich opis.

Zmiana parametru sigma przy algorytmie pierwszym (img*Gaussian → gaussian_img*Laplace → result_img) jest widoczna tym bardziej im większy rozmiar maski Gaussa ustawimy, parametr ten odpowiada za szerokość rozkładu Gaussa, czyli za to jak duże różnice będą między sąsiednimi elementami maski. Przykładowo przy zastosowaniu maski 9x9 oraz współczynnika sigma 1.2 rozmycie nie będzie duże – efekt rys. 18.



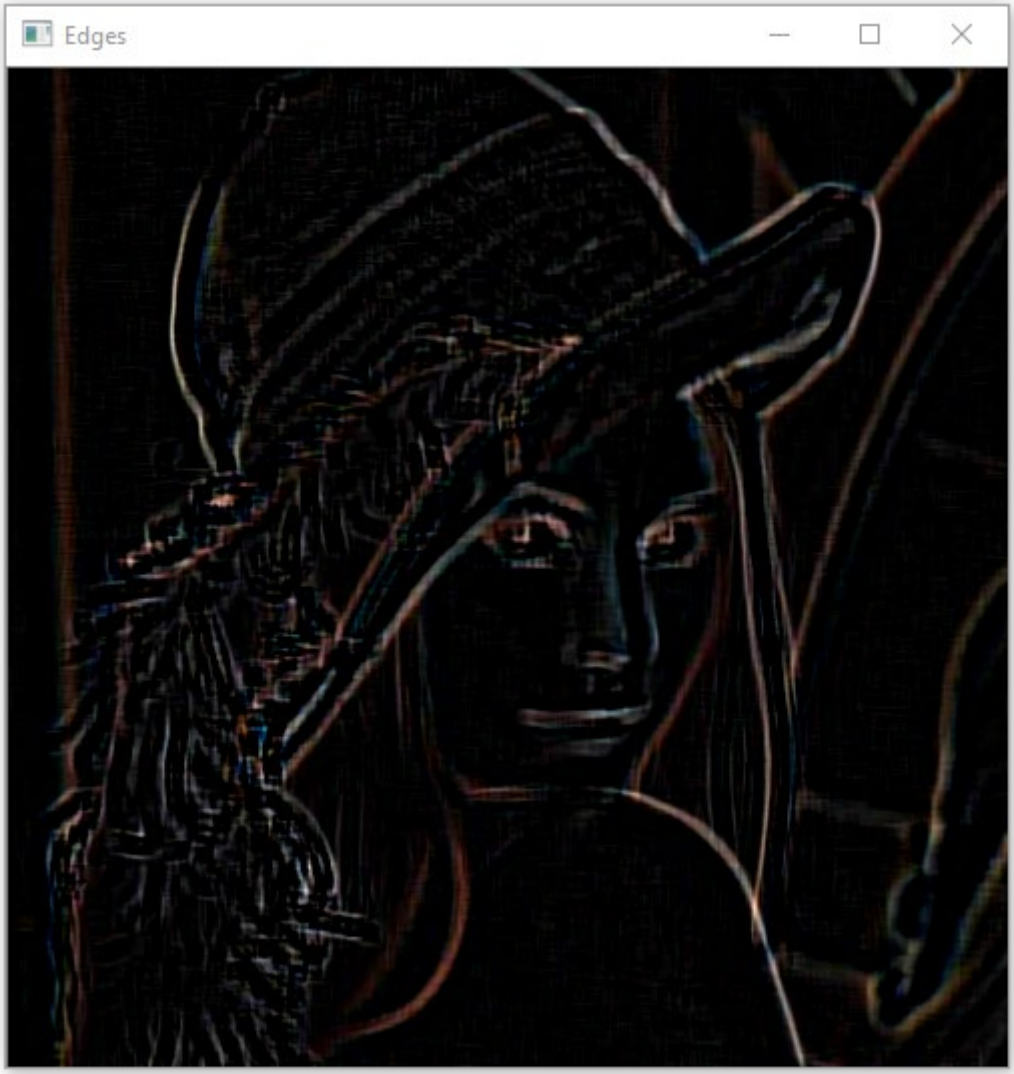
Rysunek 18

Natomiast przy tej samej wielkości maski i współczynniku sigma 8 uzyskujemy obraz jak na rys. 19.



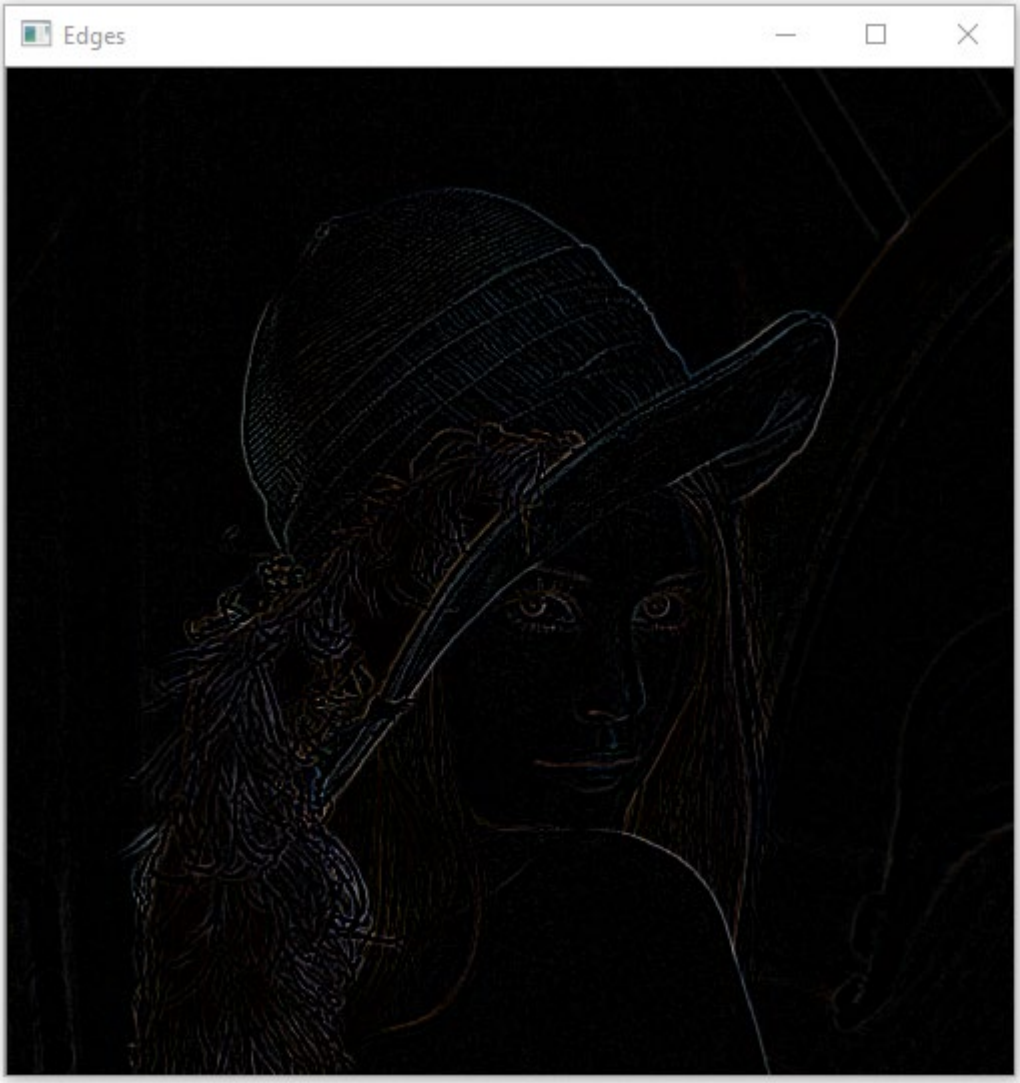
Rysunek 19

Widać, że rozmycie jest tak duże, że prawie całkowicie rozmyło nie tylko krawędzie pochodzące od szumu, ale również te pożądane. Aby uzyskać lepszy wynik można albo zmniejszyć parametr sigma lub maskę Gaussa, ale można też zwiększyć maskę Laplaca – zmiana ta sprawi, że środkowy piksel tej maski będzie zdecydowanie bardziej wzmocniony w stosunku do otoczenia. Efekt dla tych samych parametrów co poprzednio (sigma 8, Gauss 9x9) tylko z maską Laplaca 5x5 zamiast 3x3 przedstawia rys. 20.



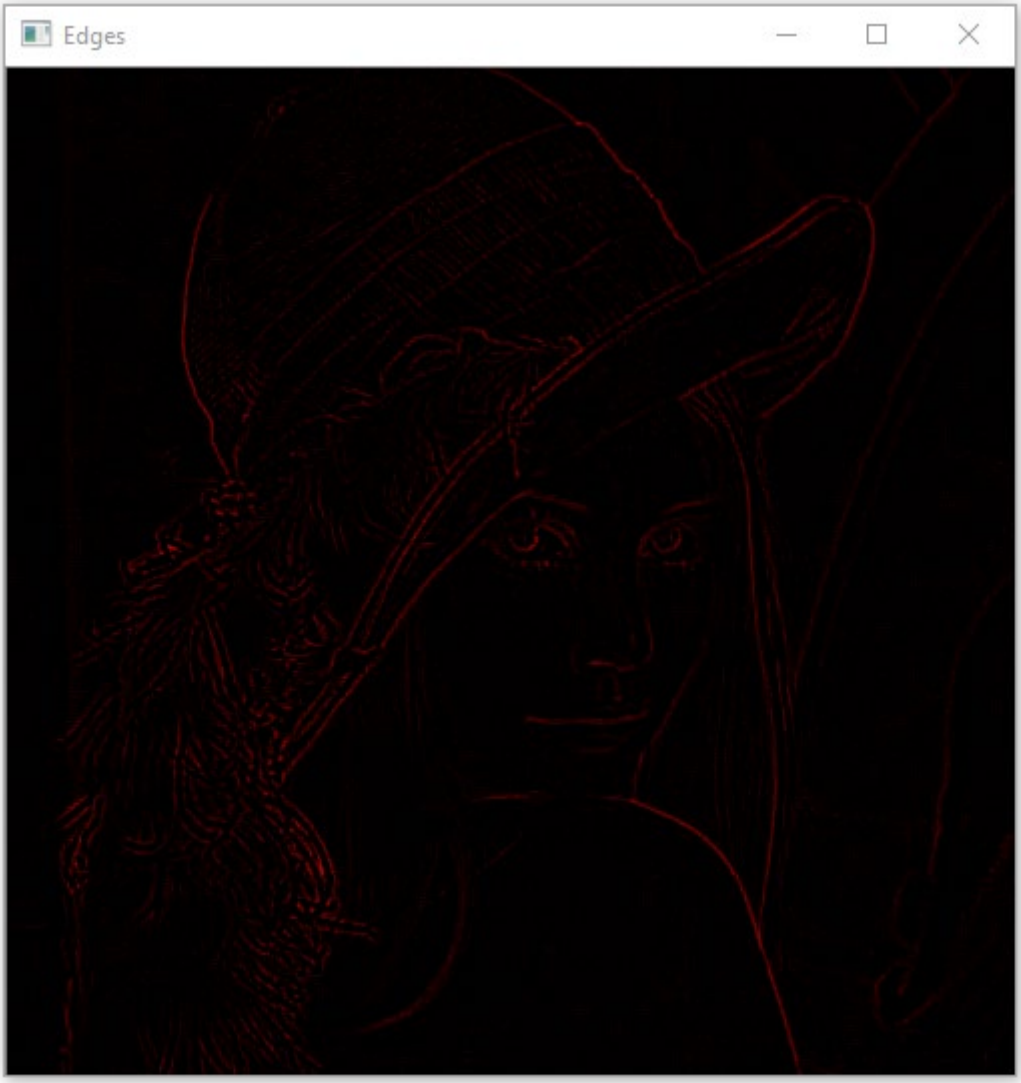
Rysunek 20

Widać, że krawędzie znów zostały odzyskane, jednak ewidentnie również widać, że rozmycie jest zbyt duże i nawet widoczne krawędzie nie są „ostre”. Warto również zauważyć, że filtry działają na konkretnej liczbie pikseli, a nie na procentowym ułamku zdjęcia, więc stosując te same parametry co powyżej dla tego samego zdjęcia, ale w znacznie wyższej rozdzielczości uzyskujemy zgoła inny efekt, również czas obliczeń programu znacząco się wydłuża – rys 21.



Rysunek 21

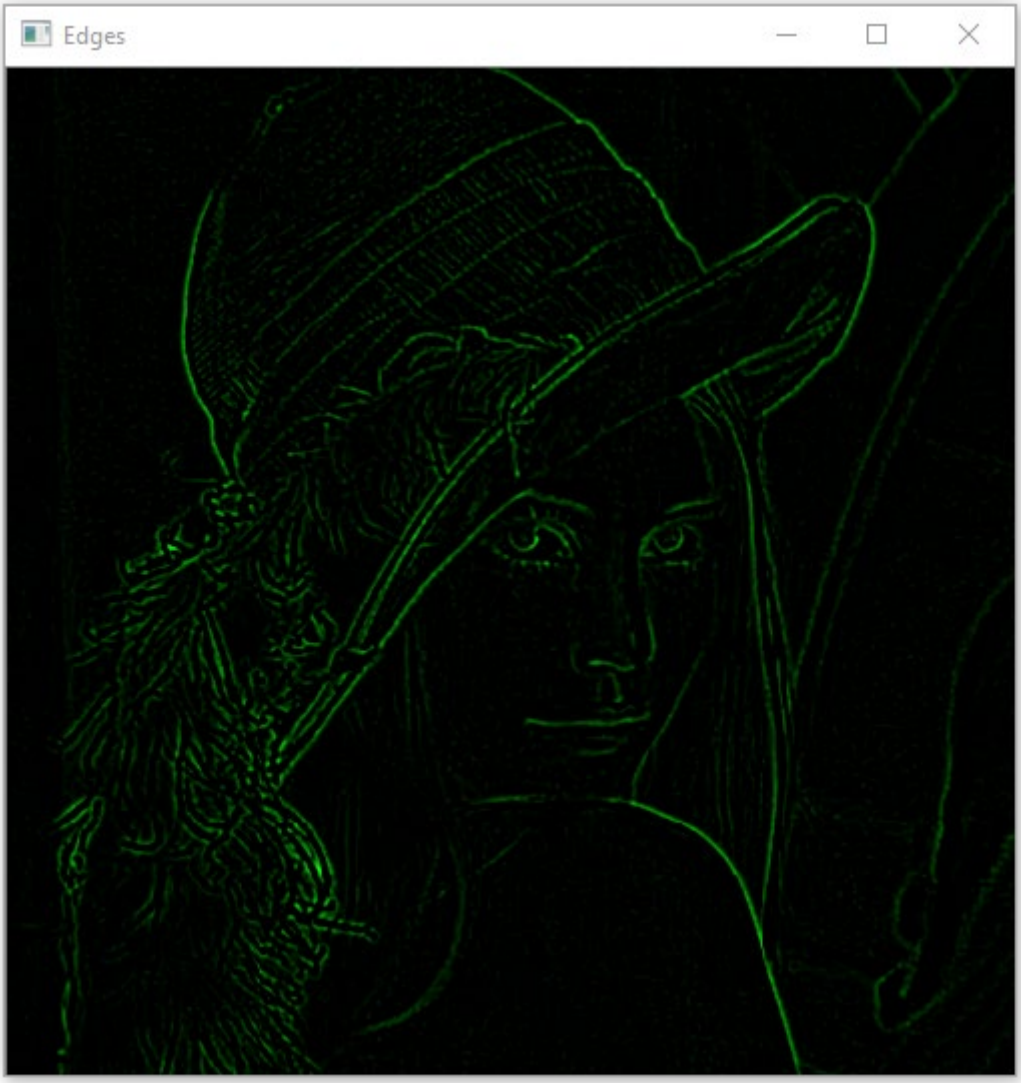
Program umożliwia również wykonywanie operacji na dowolnej kombinacji kanałów (R, G, B) zależnie od ustawienia suwaków – przykład na rys. 22 – kanał R, ponownie dla zdjęcia w niskiej rozdzielczości i masek 3x3 oraz sigma 3.



Rysunek 22

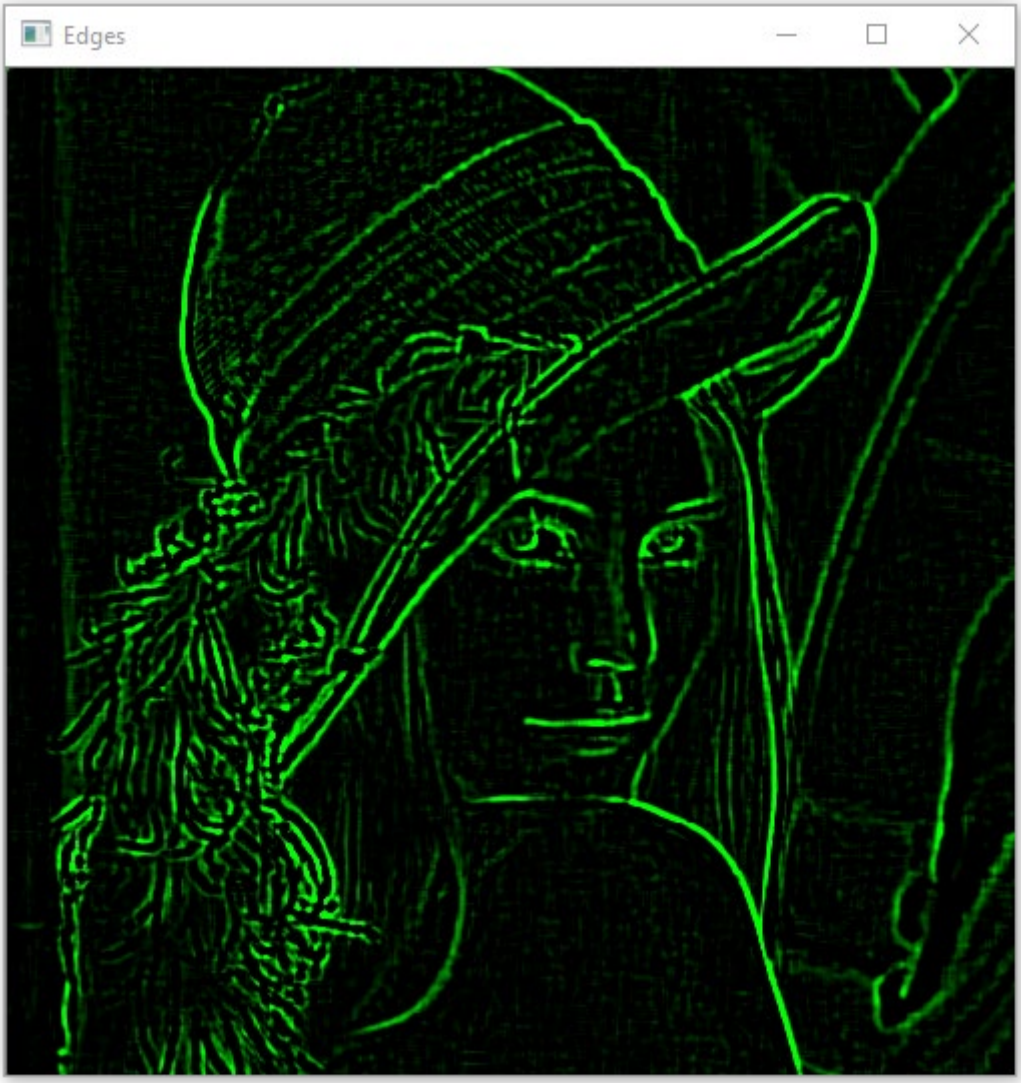
W przypadku wybrania drugiego algorytmu (LoG – pojedynczy splot) wszystkie parametry działają w taki sam sposób z wyjątkiem rozmiaru masek. W tym przypadku suwak rozmiaru maski Gaussa nie ma żadnej funkcji, natomiast suwak rozmiaru maski Laplaca odpowiada za rozmiar maski LoG – w tym przypadku suwak sigma odpowiada za: szerokość wypukłej części płaszczyzny przedstawionej na rys. 17.

Przykładowo dla wartości maski 5x5, sigma 1.2 oraz kanału G otrzymujemy efekt z rys. 23.



Rysunek 23

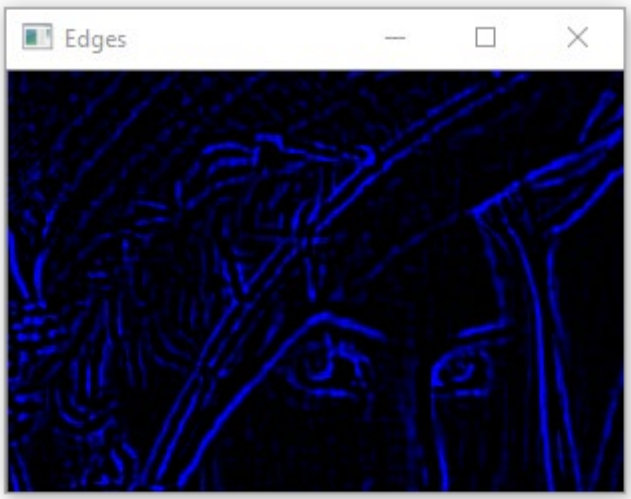
Zmiana parametru sigma na 8 powoduje zwiększenie grubości krawędzi oraz intensywności krawędzi – rys. 24.



Rysunek 24

Ostatnim parametrem jaki można modyfikować w programie jest ROI (region of interest), czyli za pomocą suwaków X1, Y1 ustawiany jest górny lewy punkt nowego obrazu wg współrzędnych obrazu źródłowego a za pomocą X2 i Y2 dolny prawy punkt nowego obrazu.

Przykładowy wynik operacji dla algorytmu (LoG – jeden splot) – kanał B, X1 = 100, Y1 = 100, X2 = 420, Y2 = 300, sigma 8, rozmiar maski 5x5, rozdzielczość wyjściowego obrazu 512x512 – rys. 25.



Rysunek 25

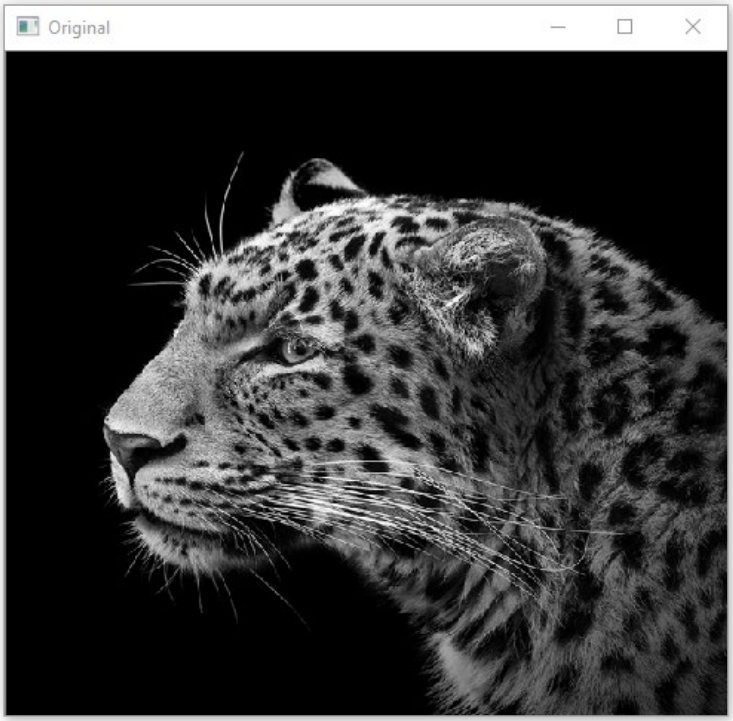
W przypadku operacji na obrazie jednokanałowym (w skali szarości) suwaki odpowiedzialne za kanały RGB nie spełniają żadnej funkcji. Należy jednak zaznaczyć, że OpenCV w przypadku standardowego ładowania plików każdy obraz automatycznie jest konwertowany na 3 kanałowy:

```
loaded_img = imread("lena1.jpg");
```

Aby faktycznie pracować na obrazie jednokanałowym należy przy wczytywaniu pliku zastosować flagę: CV_LOAD_IMAGE_GRAYSCALE:

```
loaded_img = imread("cheetah.jpg", CV_LOAD_IMAGE_GRAYSCALE);
```

Efekt działania na obrazie jednokanałowym z domyślnymi ustawieniami parametrów przedstawiają rysunki 26 (źródłowy) oraz 27 (efekt).



Rysunek 26

Rysunek 27

6. Bibliografia

[1] https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

[2] https://en.wikipedia.org/wiki/Gaussian_filter

[3] https://en.wikipedia.org/wiki/Gaussian_blur

[4] https://en.wikipedia.org/wiki/Mexican_hat_wavelet

[5] machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html