Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Отчет

По дисциплине: “Обработка изображений”

Лабораторная работа №№1-4

Выполнил:

ст.гр. ИС/б-32

Долженко И.А.

Проверила:

Сырых О.А.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Разработать интерфейс программного модуля для работы с растровым изображением. Изучить базовые методы работы с изображением и алгоритмы преобразования видов изображений. Изучить алгоритмы построения гистограммы, изменения яркости и контрастности изображения. Изучить алгоритмы наложения шумов и алгоритмы шумоподавления на растровых изображениях. Изучить алгоритмы фильтрации изображения с целью выделения границ.

2 ХОД РАБОТЫ

1. Разработаем интерфейс программного модуля.

Модуль содержит:

* окно для изображения с переключением на исходное и обработанное;
* верхнее меню;
* строку состояния;
* панель инструментов.

Модуль обеспечивает:

* загрузку и сохранение изображения;
* просмотр информации о файле;
* навигацию по истории обработанных изображений.

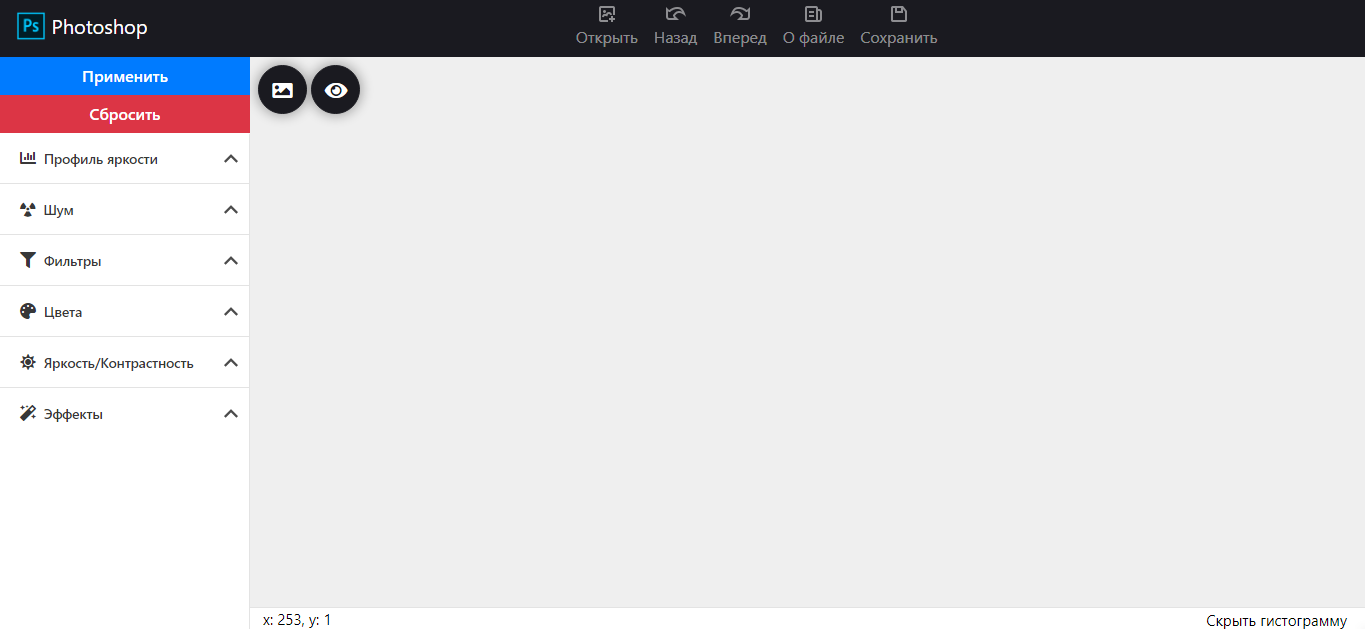


Рисунок 1 – Разработанный пользовательский интерфейс

2. В программный модуль для обработки изображения добавим инструментальную панель Информация, содержащую следующие данные:

- название файла;

- формат файла;

- размер файла;

- ширину, высоту файла.

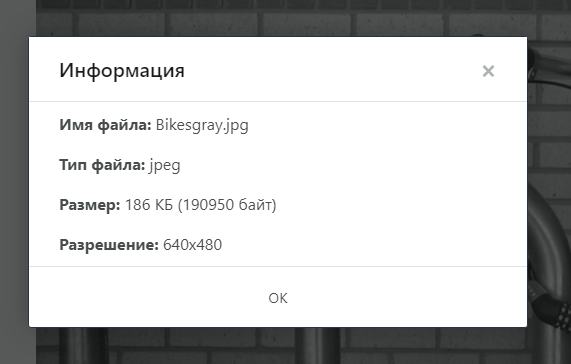


Рисунок 2 – Панель Информация о файле

3. Добавим в программный модуль функцию, позволяющую проводить преобразование изображения в оттенки серого.

Алгоритм преобразования к оттенкам серого заключается в получении средней яркости всех каналов и последующего копирования данного значения во все три канала (R=G=B=Среднее значение).

Фрагмент кода, реализующий данный алгоритм:

for (let i = 0; i < imgPixels.data.length; i += 4) {

let avg = (imgPixels.data[i] + imgPixels.data[i + 1] + imgPixels.data[i + 2]) / 3;

imgPixels.data[i] = avg;

imgPixels.data[i + 1] = avg;

imgPixels.data[i + 2] = avg;

}



Рисунок 3 – Результат преобразования к оттенкам серого

4. Добавим в программный модуль функцию, позволяющую проводить преобразование изображения в негатив.

Негатив получается простой заменой значения каждого канала на его дополнение до 255 (например, R = 255 – R).

Фрагмент кода, реализующий данный алгоритм:

for (let i = 0; i < imgPixels.data.length; i += 4) {

imgPixels.data[i] = 255 - imgPixels.data[i];

imgPixels.data[i + 1] = 255 - imgPixels.data[i + 1];

imgPixels.data[i + 2] = 255 - imgPixels.data[i + 2];

}



Рисунок 4 – Результат преобразования к негативу

5. Добавим в программный модуль функцию, позволяющую проводить преобразование изображения в бинарное.

Для преобразования изображения в двуцветное выбирается некий порог (в нашем случае порог определяется диапазоном значения от 0 до 255) все значения ниже которого превращаются в один цвет, а выше – в другой цвет (цвета определяются пользователем).

Фрагмент кода, реализующий данный алгоритм:

for (let i = 0; i < imgPixels.data.length; i += 4) {

let v = (imgPixels.data[i] + imgPixels.data[i + 1] + imgPixels.data[i + 2]) / 3;

imgPixels.data[i] = v > binMul ? color1.r : color2.r;

imgPixels.data[i + 1] = v > binMul ? color1.g : color2.g;

imgPixels.data[i + 2] = v > binMul ? color1.b : color2.b;

}



Рисунок 5 – Результат преобразования в бинарное

6. Добавим в программный модуль функцию, позволяющую проводить изменение яркости изображения.

Повышение/снижение яркости – это, соответственно, сложение/вычитание значения каждого канала с некоторым фиксированным значением (также в пределах от 0 до 255).

Фрагмент кода, реализующий данный алгоритм:

for (let i = 0; i < imgPixels.data.length; i += 4) {

imgPixels.data[i] += brightnessMul;

imgPixels.data[i + 1] += brightnessMul;

imgPixels.data[i + 2] += brightnessMul;

}

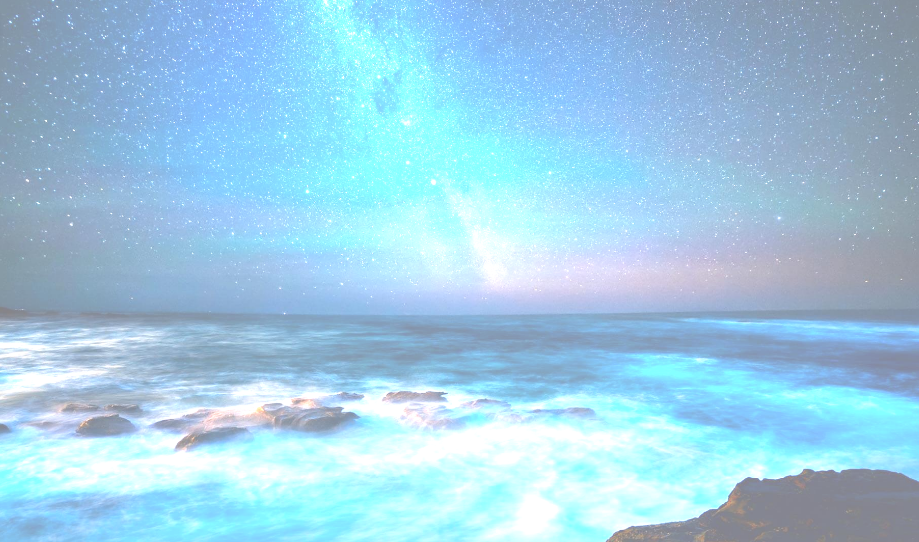


Рисунок 6 – Результат увеличения яркости изображения

7. Добавим в программный модуль функцию, позволяющую проводить изменение контрастности изображения.

Повышение/снижение контрастности – это, соответственно, умножение/деление значения каждого канала на некоторое фиксированное значение.

Фрагмент кода, реализующий данный алгоритм:

contrastMul = (contrastMul / 100) + 1;

let intercept = 128 \* (1 - contrastMul);

for (let i = 0; i < imgPixels.data.length; i += 4) {

imgPixels.data[i] = imgPixels.data[i] \* contrastMul + intercept;

imgPixels.data[i + 1] = imgPixels.data[i + 1] \* contrastMul + intercept;

imgPixels.data[i + 2] = imgPixels.data[i + 2] \* contrastMul + intercept;

}



Рисунок 7 – Результат увеличения контрастности изображения

8. Добавим в программный модуль функцию, позволяющую проводить построение гистограмм изображений по яркости и трем цветовым каналам.

Для построения гистограммы необходимо загрузить любое доступное изображение, после чего пройтись по каждому пикселю и рассчитать его яркость, полагая, что она лежит в интервале от 0 до 255. После расчета яркостей нужно посчитать количество каждого вычисленного значения яркости.

Фрагмент кода, реализующий данный алгоритм:

for (let i = 0; i < iD.length; i += 4) {

pixelSum[Math.round((iD[i] + iD[i + 1] + iD[i + 2]) / 3)]++;

r[iD[i]]++;

g[iD[i + 1]]++;

b[iD[i + 2]]++;

}

drawHistogram(pixelSum, r, g, b);

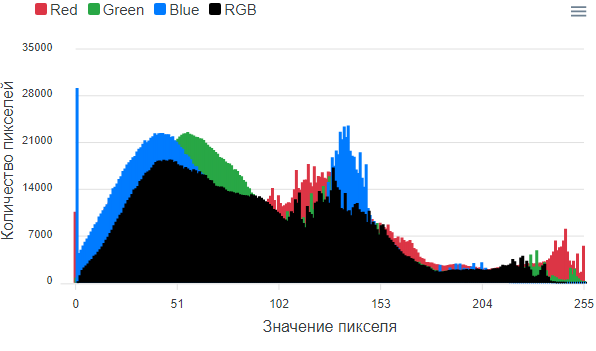


Рисунок 8 – Результат построения гистограммы

9. Добавим в программный модуль функцию, позволяющую проводить построение профиля яркости по трем цветовым каналам. Построение профиля может проводиться как по вертикальное, так и по горизонтальной оси.

По вертикальной оси профиля отсчетов яркости изображения откладываются значения яркости пикселей анализируемого среза, по горизонтальной оси – длина профиля в пикселях.

Фрагмент кода, реализующий данный алгоритм:

if (horizontal) {

iD = ctx.getImageData(0, yClick, canvas.width, 1).data;

} else {

iD = ctx.getImageData(xClick, 0, 1, canvas.height).data;

}

for (let i = 0; i < iD.length; i += 4) {

r.push(iD[i]);

g.push(iD[i + 1]);

b.push(iD[i + 2]);

}

drawProfile(r, g, b);



Рисунок 9 – Результат построения профиля яркости

10. Добавим в программный модуль функцию, позволяющую проводить наложение шумов на изображение.

Шум "Соль-перец" накладывается непосредственно на черно-белое изображение, состоящее только из белого и черных символов. При этом с заданной вероятностью по случайным координатам на изображении выбирается пиксель и его цвет меняется на противоположный (черный цвет заменяется белым, белый - черным).

Фрагмент кода, реализующий данный алгоритм:

for (let i = 0; i < imgPixels.data.length; i += 4) {

let isBlack = imgPixels.data[i] == 0 ? true : false;

let rand = Math.random();

imgPixels.data[i] = isBlack ? (rand >= noiseMul ? 0 : 255) : (rand >= noiseMul ? 255 : 0);

imgPixels.data[i + 1] = isBlack ? (rand >= noiseMul ? 0 : 255) : (rand >= noiseMul ? 255 : 0);

imgPixels.data[i + 2] = isBlack ? (rand >= noiseMul ? 0 : 255) : (rand >= noiseMul ? 255 : 0);

}



Рисунок 10 – Результат применения шума

11. Добавим в программный модуль функцию, позволяющую проводить фильтрацию изображения линейным сглаживающим фильтром.

Сглаживающий фильтр основывается на следующем принципе: находится среднее арифметическое значение всех элементов рабочего окна изображения отдельно по каждому из каналов), после чего это среднее значение становится значением среднего элемента.

Фрагмент кода, реализующий данный алгоритм:

for (let j = 0; j < canvas.width; j++) {

for (let i = 0; i < canvas.height; i++) {

if (i == 0 || j == 0 || i + 1 == canvas.height || j + 1 == canvas.width) { }

else {

let r\_summ = temp[i - 1][j - 1][0];

let g\_summ = temp[i - 1][j - 1][1];

let b\_summ = temp[i - 1][j - 1][2];

r\_summ = r\_summ + temp[i - 1][j][0];

g\_summ = g\_summ + temp[i - 1][j][1];

b\_summ = b\_summ + temp[i - 1][j][2];

r\_summ += temp[i - 1][j + 1][0];

g\_summ += temp[i - 1][j + 1][1];

b\_summ += temp[i - 1][j + 1][2];

r\_summ = r\_summ + temp[i][j - 1][0];

g\_summ = g\_summ + temp[i][j - 1][1];

b\_summ = b\_summ + temp[i][j - 1][2];

r\_summ = r\_summ + temp[i][j][0];

g\_summ = g\_summ + temp[i][j][1];

b\_summ = b\_summ + temp[i][j][2];

r\_summ = r\_summ + temp[i][j + 1][0];

g\_summ = g\_summ + temp[i][j + 1][1];

b\_summ = b\_summ + temp[i][j + 1][2];

r\_summ = r\_summ + temp[i + 1][j - 1][0];

g\_summ = g\_summ + temp[i + 1][j - 1][1];

b\_summ = b\_summ + temp[i + 1][j - 1][2];

r\_summ = r\_summ + temp[i + 1][j][0];

g\_summ = g\_summ + temp[i + 1][j][1];

b\_summ = b\_summ + temp[i + 1][j][2];

r\_summ = r\_summ + temp[i + 1][j + 1][0];

g\_summ = g\_summ + temp[i + 1][j + 1][1];

b\_summ = b\_summ + temp[i + 1][j + 1][2];

r\_summ = r\_summ / 9;

g\_summ = g\_summ / 9;

b\_summ = b\_summ / 9;

temp[i][j][0] = Math.floor(r\_summ);

temp[i][j][1] = Math.floor(g\_summ);

temp[i][j][2] = Math.floor(b\_summ);

}

}

}



Рисунок 11 – Результат применения линейного сглаживающего фильтра

12. Добавим в программный модуль функцию, позволяющую проводить фильтрацию изображения нелинейным медианным фильтром.

Нелинейный медианный фильтр основывается на нахождении медианы – среднего элемента последовательности в результате её упорядочения по возрастанию/убыванию и присваиванию найденного значения только среднему элементу.

Фрагмент кода, реализующий данный алгоритм:

for (let j = 0; j < canvas.width; j++) {

for (let i = 0; i < canvas.height; i++) {

if (i == 0 || j == 0 || i + 1 == canvas.height || j + 1 == canvas.width) { }

else {

let r\_summ = [

temp[i - 1][j - 1][0],

temp[i - 1][j][0],

temp[i - 1][j + 1][0],

temp[i][j - 1][0],

temp[i][j][0],

temp[i][j + 1][0],

temp[i + 1][j - 1][0],

temp[i + 1][j][0],

temp[i + 1][j + 1][0]

];

let g\_summ = [

temp[i - 1][j - 1][1],

temp[i - 1][j][1],

temp[i - 1][j + 1][1],

temp[i][j - 1][1],

temp[i][j][1],

temp[i][j + 1][1],

temp[i + 1][j - 1][1],

temp[i + 1][j][1],

temp[i + 1][j + 1][1]

];

let b\_summ = [

temp[i - 1][j - 1][2],

temp[i - 1][j][2],

temp[i - 1][j + 1][2],

temp[i][j - 1][2],

temp[i][j][2],

temp[i][j + 1][2],

temp[i + 1][j - 1][2],

temp[i + 1][j][2],

temp[i + 1][j + 1][2]

];

r\_summ.sort();

g\_summ.sort();

b\_summ.sort();

temp[i][j][0] = r\_summ[4];

temp[i][j][1] = g\_summ[4];

temp[i][j][2] = b\_summ[4];

}

}

}



Рисунок 12 – Результат применения нелинейного медианного фильтра

13. Добавим в программный модуль фильтры выделения границ следующими методами:

1) *Метод Лапласа* осуществляет домножение каждого элемента двумерной апертуры 3×3 на соответствующий элемент матрицы Лапласа:

Фрагмент кода, реализующий данный метод:

let imgPixels = ctx.getImageData(0, 0, canvas.width, canvas.height);

let weight = [

-1, -2, -1,

-2, 12, -2,

-1, -2, -1

]

imgPixels = convolution(imgPixels, weight)

ctx.putImageData(imgPixels, 0, 0, 0, 0, imgPixels.width, imgPixels.height);



Рисунок 13 – Результат применения метода Лапласа

2) *Метод Робертса* работает с двумерной апертурой 2×2. Дифференцирование производится с помощью выражения:

Окончательное значение G заносится в элемент , после чего рабочее окно сдвигается на один элемент влево.

Фрагмент кода, реализующий данный метод:

for (let j = 0; j < canvas.width; j++) {

for (let i = 0; i < canvas.height; i++) {

if (i == 0 || j == 0 || i + 1 == canvas.height || j + 1 == canvas.width) { }

else {

let r = Math.abs(temp[i][j][0] - temp[i + 1][j + 1][0]) + Math.abs(temp[i][j + 1][0] - temp[i + 1][j][0]);

let g = Math.abs(temp[i][j][1] - temp[i + 1][j + 1][1]) + Math.abs(temp[i][j + 1][1] - temp[i + 1][j][1]);

let b = Math.abs(temp[i][j][2] - temp[i + 1][j + 1][2]) + Math.abs(temp[i][j + 1][2] - temp[i + 1][j][2]);

temp[i][j][0] = r;

temp[i][j][1] = g;

temp[i][j][2] = b;

}

}

}



Рисунок 14 – Результат применения метода Робертса

3) *Метод Кирша*

Сначала в цикле находятся все значения переменных и , где i изменяется от 0 до 7:

После находятся значения модуля разности для каждого i от 0 до 7 и значение максимума среди этих модулей. Окончательное значение заносится в элемент F, после чего рабочее окно сдвигается на один элемент влево.

Фрагмент кода, реализующий данный метод:

let imgPixels = ctx.getImageData(0, 0, canvas.width, canvas.height);

let weight = [

-3, 5, 5,

-3, 0, 5,

-3, -3, -3

];

imgPixels = convolution(imgPixels, weight);

ctx.putImageData(imgPixels, 0, 0, 0, 0, imgPixels.width, imgPixels.height);

function convolution(pixels, weights) {

let side = Math.round(Math.sqrt(weights.length));

let halfSide = Math.floor(side / 2);

let src = pixels.data;

let canvasWidth = pixels.width;

let canvasHeight = pixels.height;

let temporaryCanvas = document.createElement('canvas');

let temporaryCtx = temporaryCanvas.getContext('2d');

let outputData = temporaryCtx.createImageData(canvasWidth, canvasHeight);

for (let y = 0; y < canvasHeight; y++) {

for (let x = 0; x < canvasWidth; x++) {

let dstOff = (y \* canvasWidth + x) \* 4;

let sumReds = 0, sumGreens = 0, sumBlues = 0;

for (let kernelY = 0; kernelY < side; kernelY++) {

for (let kernelX = 0; kernelX < side; kernelX++) {

let currentKernelY = y + kernelY - halfSide;

let currentKernelX = x + kernelX - halfSide;

if (currentKernelY >= 0 &&

currentKernelY < canvasHeight &&

currentKernelX >= 0 &&

currentKernelX < canvasWidth) {

let offset = (currentKernelY \* canvasWidth + currentKernelX) \* 4;

let weight = weights[kernelY \* side + kernelX];

sumReds += src[offset] \* weight;

sumGreens += src[offset + 1] \* weight;

sumBlues += src[offset + 2] \* weight;

}

}

}

outputData.data[dstOff] = sumReds;

outputData.data[dstOff + 1] = sumGreens;

outputData.data[dstOff + 2] = sumBlues;

outputData.data[dstOff + 3] = 255;

}

}

return outputData;

}



Рисунок 15 – Результат применения метода Кирша

4) *Метод Собела*

Центральному ()-му пикселю вместо присваивается значение яркости:

,

где ,

Фрагмент кода, реализующий данный метод:

for (let i = 0; i < canvas.height; i++) {

for (let j = 0; j < canvas.width; j++) {

if (i == 0 || j == 0 || i + 1 == canvas.height || j + 1 == canvas.width) { }

else {

let x\_r =

(temp[i - 1][j + 1][0] + 2 \* temp[i][j + 1][0] + temp[i + 1][j + 1][0]) -

(temp[i - 1][j - 1][0] + 2 \* temp[i][j - 1][0] + temp[i + 1][j - 1][0]);

let y\_r =

(temp[i - 1][j - 1][0] + 2 \* temp[i - 1][j][0] + temp[i - 1][j + 1][0]) -

(temp[i + 1][j - 1][0] + 2 \* temp[i + 1][j][0] + temp[i + 1][j + 1][0]);

let x\_g =

(temp[i - 1][j + 1][1] + 2 \* temp[i][j + 1][1] + temp[i + 1][j + 1][1]) -

(temp[i - 1][j - 1][1] + 2 \* temp[i][j - 1][1] + temp[i + 1][j - 1][1]);

let y\_g =

(temp[i - 1][j - 1][1] + 2 \* temp[i - 1][j][1] + temp[i - 1][j + 1][1]) -

(temp[i + 1][j - 1][1] + 2 \* temp[i + 1][j][1] + temp[i + 1][j + 1][1]);

let x\_b =

(temp[i - 1][j + 1][2] + 2 \* temp[i][j + 1][2] + temp[i + 1][j + 1][2]) -

(temp[i - 1][j - 1][2] + 2 \* temp[i][j - 1][2] + temp[i + 1][j - 1][2]);

let y\_b =

(temp[i - 1][j - 1][2] + 2 \* temp[i - 1][j][2] + temp[i - 1][j + 1][2]) -

(temp[i + 1][j - 1][2] + 2 \* temp[i + 1][j][2] + temp[i + 1][j + 1][2]);

let g\_r = Math.sqrt(x\_r \* x\_r + y\_r \* y\_r);

let g\_g = Math.sqrt(x\_g \* x\_g + y\_g \* y\_g);

let g\_b = Math.sqrt(x\_b \* x\_b + y\_b \* y\_b);

outputData[i][j][0] = g\_r;

outputData[i][j][1] = g\_g;

outputData[i][j][2] = g\_b;

outputData[i][j][3] = 255;

}

}

}



Рисунок 16 – Результат применения метода Собела

5) *Метод Уоллеса*

Сразу находится новое значение центрального элемента по формуле:

Фрагмент кода, реализующий данный метод:

for (let i = 0; i < canvas.height; i++) {

for (let j = 0; j < canvas.width; j++) {

if (i == 0 || j == 0 || i + 1 == canvas.height || j + 1 == canvas.width) { }

else {

let f\_r = Math.log((

((temp[i][j][0] + 1) / (temp[i - 1][j][0] + 1)) \*

((temp[i][j][0] + 1) / (temp[i][j + 1][0] + 1)) \*

((temp[i][j][0] + 1) / (temp[i + 1][j][0] + 1)) \*

((temp[i][j][0] + 1) / (temp[i][j - 1][0] + 1))

)) / 4;

let f\_g = Math.log((

((temp[i][j][1] + 1) / (temp[i - 1][j][1] + 1)) \*

((temp[i][j][1] + 1) / (temp[i][j + 1][1] + 1)) \*

((temp[i][j][1] + 1) / (temp[i + 1][j][1] + 1)) \*

((temp[i][j][1] + 1) / (temp[i][j - 1][1] + 1))

)) / 4;

let f\_b = Math.log((

((temp[i][j][2] + 1) / (temp[i - 1][j][2] + 1)) \*

((temp[i][j][2] + 1) / (temp[i][j + 1][2] + 1)) \*

((temp[i][j][2] + 1) / (temp[i + 1][j][2] + 1)) \*

((temp[i][j][2] + 1) / (temp[i][j - 1][2] + 1))

)) / 4;

outputData[i][j][0] = f\_r \* 1000;

outputData[i][j][1] = f\_g \* 1000;

outputData[i][j][2] = f\_b \* 1000;

outputData[i][j][3] = 255;

}

}

}



Рисунок 17 – Результат применения метода Уоллеса

6) *Статический метод*

На первом этапе вычисляется среднее значение яркости по текущему рабочему окну:

Далее вычисляется значение среднеквадратичного отклонения значений элементов рабочего окна от среднеарифметического значения:

Затем значения всех элементов рабочего окна домножаются на полученное отклонение:

Фрагмент кода, реализующий данный метод:

for (let j = 0; j < canvas.width; j++) {

for (let i = 0; i < canvas.height; i++) {

if (i == 0 || j == 0 || i + 1 == canvas.height || j + 1 == canvas.width) { }

else {

let summ1\_r = temp[i][j][0] + temp[i + 1][j][0] + temp[i][j + 1][0] + temp[i + 1][j + 1][0];

let summ1\_g = temp[i][j][1] + temp[i + 1][j][1] + temp[i][j + 1][1] + temp[i + 1][j + 1][1];

let summ1\_b = temp[i][j][2] + temp[i + 1][j][2] + temp[i][j + 1][2] + temp[i + 1][j + 1][2];

let mu\_r = 1 / 4 \* summ1\_r;

let mu\_g = 1 / 4 \* summ1\_g;

let mu\_b = 1 / 4 \* summ1\_b;

let summ2\_r = Math.pow((temp[i][j][0] - mu\_r), 2) + Math.pow((temp[i + 1][j][0] - mu\_r), 2) + Math.pow((temp[i][j + 1][0] - mu\_r), 2) + Math.pow((temp[i + 1][j + 1][0] - mu\_r), 2);

let summ2\_g = Math.pow((temp[i][j][1] - mu\_g), 2) + Math.pow((temp[i + 1][j][1] - mu\_g), 2) + Math.pow((temp[i][j + 1][1] - mu\_g), 2) + Math.pow((temp[i + 1][j + 1][1] - mu\_g), 2);

let summ2\_b = Math.pow((temp[i][j][2] - mu\_b), 2) + Math.pow((temp[i + 1][j][2] - mu\_b), 2) + Math.pow((temp[i][j + 1][2] - mu\_b), 2) + Math.pow((temp[i + 1][j + 1][2] - mu\_b), 2);

let tau\_r = Math.sqrt(1 / 4 \* summ2\_r);

let tau\_g = Math.sqrt(1 / 4 \* summ2\_g);

let tau\_b = Math.sqrt(1 / 4 \* summ2\_b);

outputData[i][j][0] = (tau\_r \* temp[i][j][0] + val);

outputData[i][j][1] = (tau\_g \* temp[i][j][1] + val);

outputData[i][j][2] = (tau\_b \* temp[i][j][2] + val);

outputData[i][j][3] = 255;

outputData[i][j + 1][0] = (tau\_r \* temp[i][j + 1][0] + val);

outputData[i][j + 1][1] = (tau\_g \* temp[i][j + 1][1] + val);

outputData[i][j + 1][2] = (tau\_b \* temp[i][j + 1][2] + val);

outputData[i][j + 1][3] = 255;

outputData[i + 1][j][0] = (tau\_r \* temp[i + 1][j][0] + val);

outputData[i + 1][j][1] = (tau\_g \* temp[i + 1][j][1] + val);

outputData[i + 1][j][2] = (tau\_b \* temp[i + 1][j][2] + val);

outputData[i + 1][j][3] = 255;

outputData[i + 1][j + 1][0] = (tau\_r \* temp[i + 1][j + 1][0] + val);

outputData[i + 1][j + 1][1] = (tau\_g \* temp[i + 1][j + 1][1] + val);

outputData[i + 1][j + 1][2] = (tau\_b \* temp[i + 1][j + 1][2] + val);

outputData[i + 1][j + 1][3] = 255;

}

}

}



Рисунок 18 – Результат применения статического метода

ВЫВОД

В ходе выполнения лабораторной работы был разработан интерфейс программного модуля для работы с растровым изображением. Были изучены базовые методы работы с изображением и алгоритмы преобразования видов изображений; алгоритмы построения гистограммы, изменения яркости и контрастности изображения; алгоритмы наложения шумов и алгоритмы шумоподавления на растровых изображениях. Созданы функции фильтрации изображения с целью выделения границ.