МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н. Э. БАУМАНА

Факультет информатики и систем управления

Кафедра теоретической информатики и компьютерных технологи

Лабораторная работа №4

по курсу «Структуры и алгоритмы обработки больших данных»

«Выделение контуров на изображении»

Выполнил:
студент группы ИУ9-21М
Беляев А. В.

Проверил:
Магазов С. С.

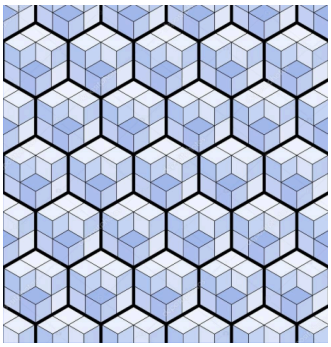
Вариант 3

Москва 2019

Цель работы

Научиться выделять границы из изображения. В качестве тестовой базы данных использовать базу данных регулярных текстур. Работать на изображениях n*10, где n- номер в списке

Вариант 3



Задание 1

Дополнить базу изображений 10 изображениями не встречающимися в базе данных регулярных текстур.

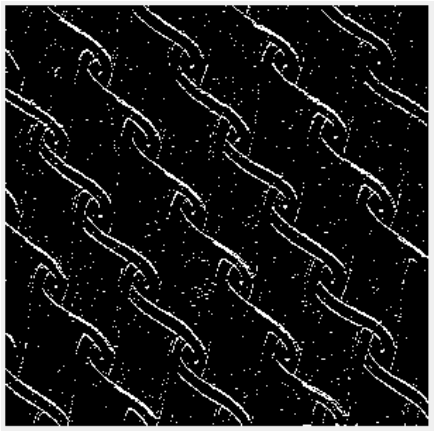
Были добавлены 10 регулярных текстур в базу текстур. Одна из них (на странице слева) была добавлена в тестовую подборку изображений.

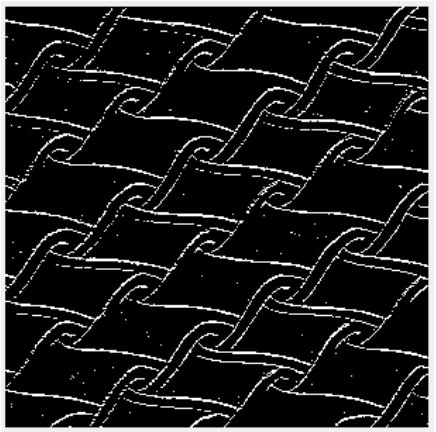
Задание 2

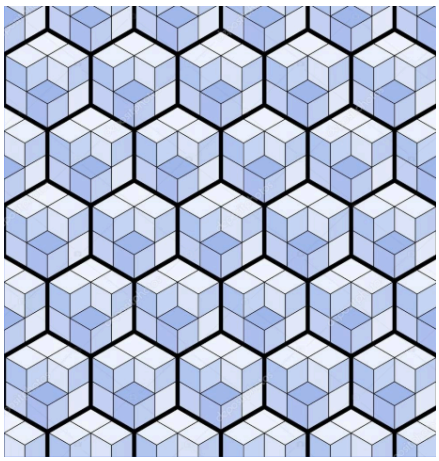
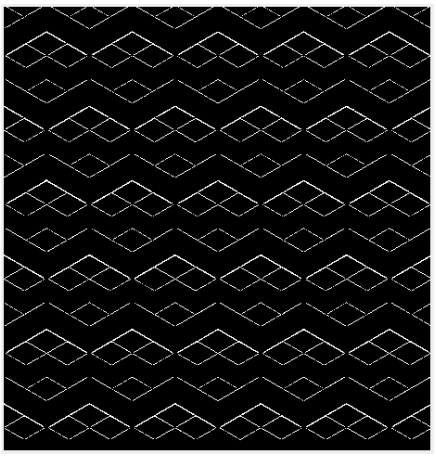
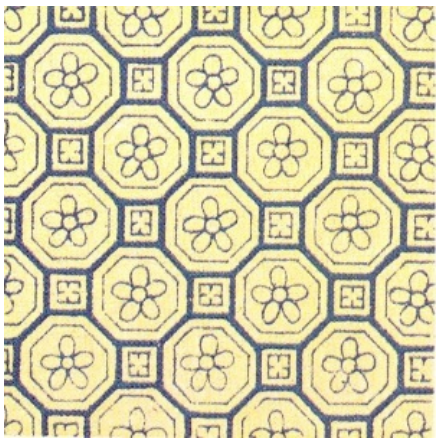
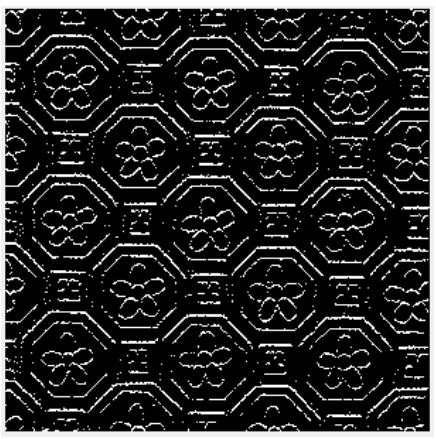
1. Написать программы, выделяющие контуры следующими методами:
    1. Дифференциальный метод
    2. Пороговый метод (Thresholding)
    3. Скелетонизация (Skeletonization)
    4. Метод локальных максимумов (NMS)
    5. Статистический метод

a. Дифференциальный метод

```
clear all;

img = imread('regular_31.jpg');
img = rgb2gray(img);

D = diff(img);
imshow(D, []);
```



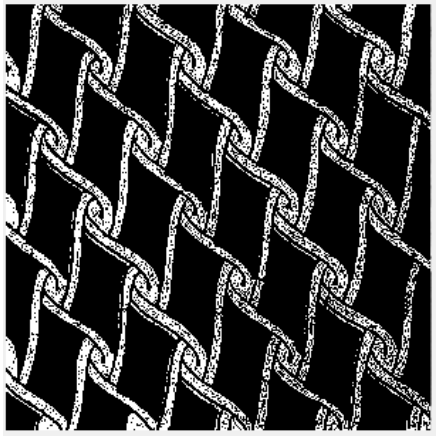



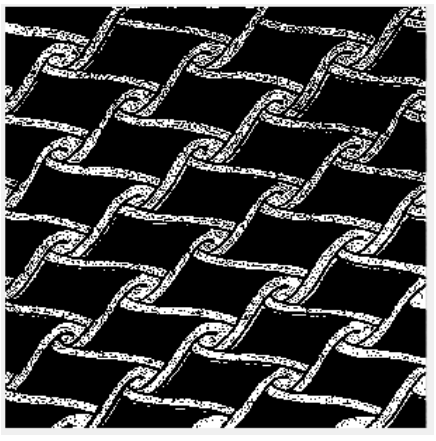






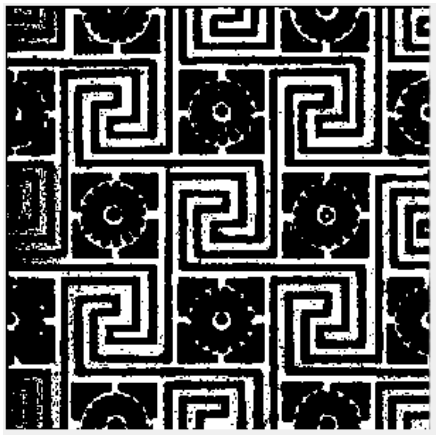
b. Пороговый метод Thresholding

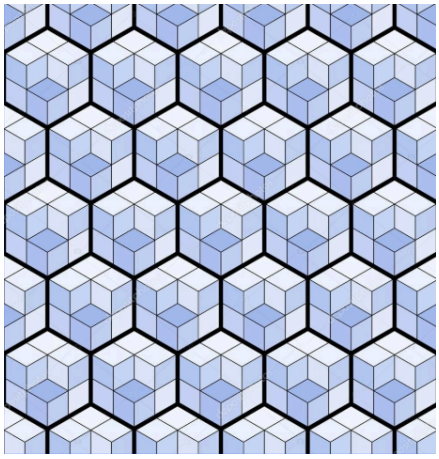
```
clear all;

BACKGROUND = 0;
CONTOUR = 255;
% own value for each image
THRESHOLD = 215;

img = imread('regular_31.jpg');
img = rgb2gray(img);
[img_h, img_w, dim] = size(img);

for i = 1:img_h
    for j = 1:img_w-1
        if img(i, j) > THRESHOLD
            img(i, j) = CONTOUR;
        else
            img(i, j) = BACKGROUND;
        end
    end
end
imshow(img);
```

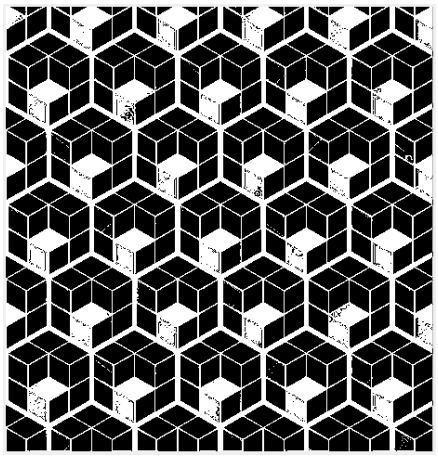
Недостатком этого метода является необходимость подбирать свой параметр THRESHOLD для каждого изображения.

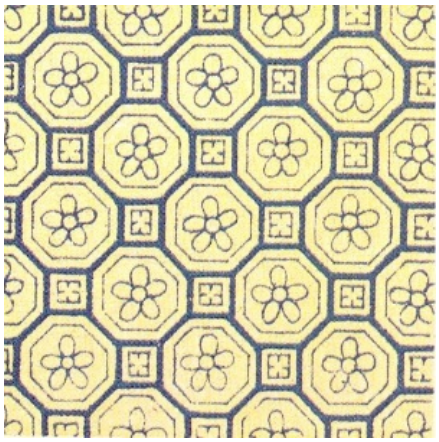


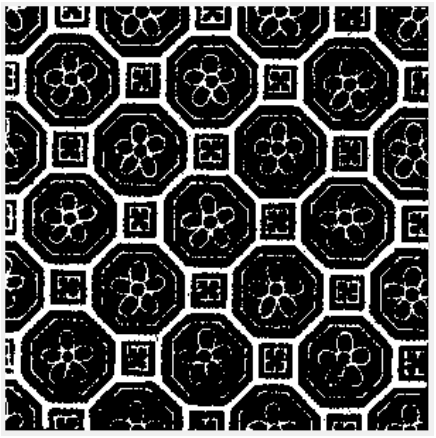






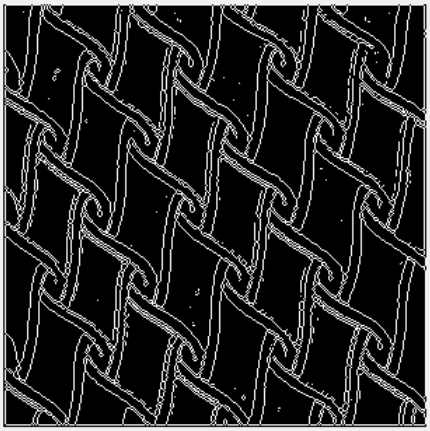


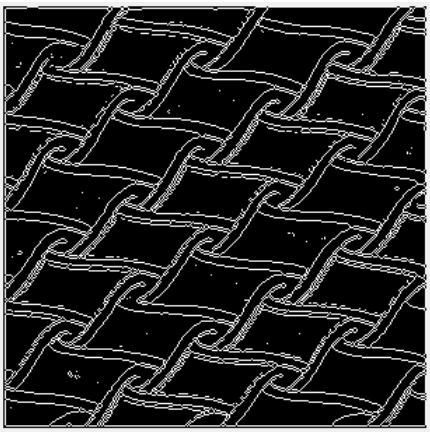


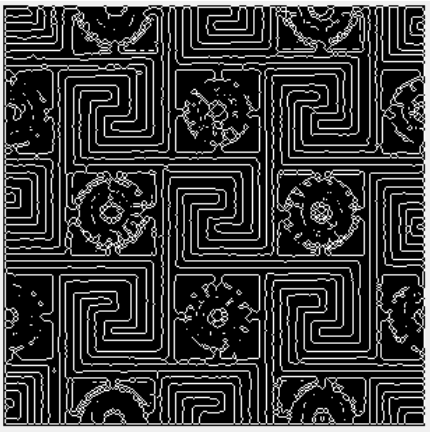


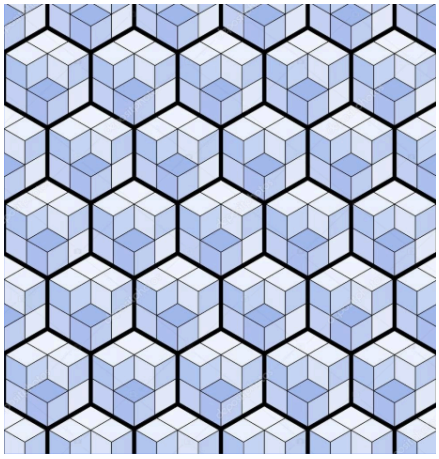
c. Скелетонизация Skeletonization

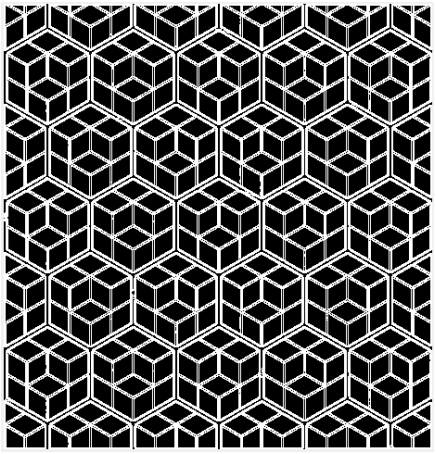
```
clear all;

img = imread('regular_31.jpg');
img = rgb2gray(img);
img = double (img);

sobelm_x = [ 1 2 1; 0 0 0; -1 -2 -1];
sobelm_y = [ -1 0 1; -2 0 2; -1 0 1];
% convolute with sobel kernel
w_x = conv2(img, sobelm_x);
w_y = conv2(img, sobelm_y);
w = sqrt(double(w_x.^2 + w_y.^2));

imwrite(w, gray(256), 'conv.png');
img = imread('conv.png');
binarized = imbinarize(img);
skeletonized = bwskel(binarized);
imshow(skeletonized);
```

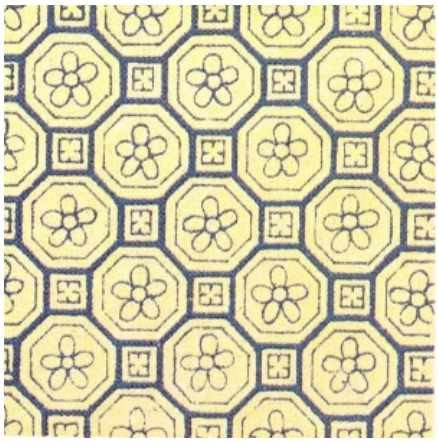


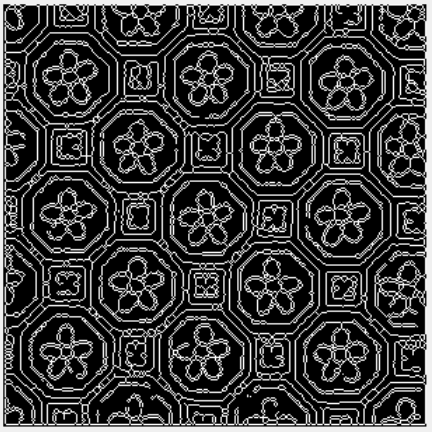












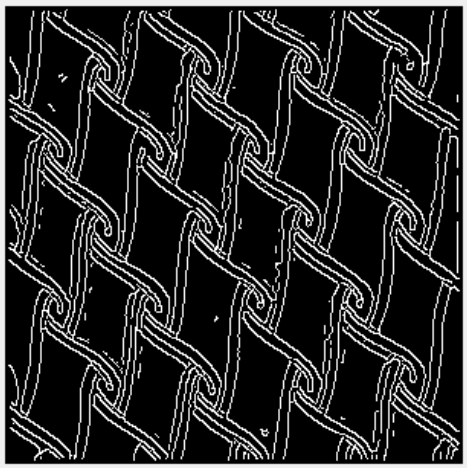
d. Метод локальных максимумов (NMS)

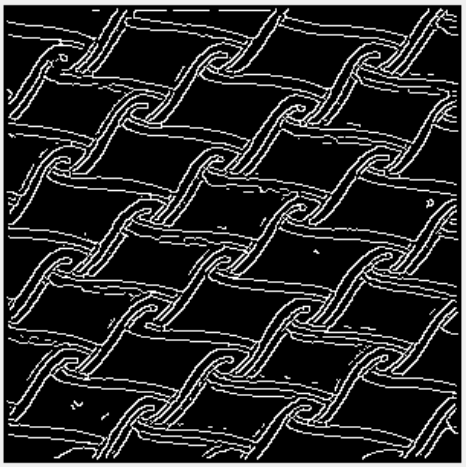
Ниже приведен код, имеющийся в библиотеке Matlab Mathworks, использующий в том числе 8-компонентное кодирование для выделения границы

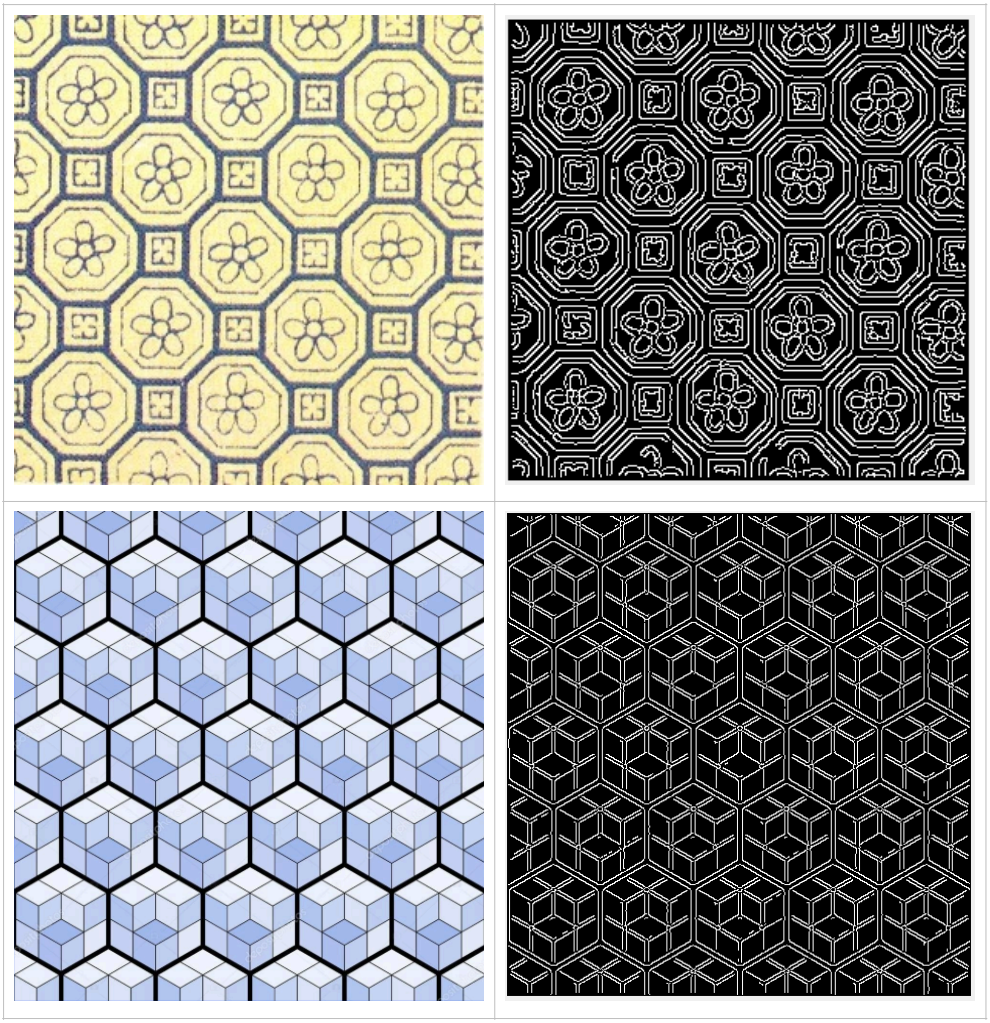
```
clear all;
clc;
%Input image
img = imread ('regular_39.jpg');
img = rgb2gray(img);
img = double (img);
%Value for Thresholding
T_Low = 0.075;
T_High = 0.175;
%Gaussian Filter Coefficient
B = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9, 12, 9, 4;2, 4, 5, 4, 2 ];
B = 1/159.* B;
%Convolution of image by Gaussian Coefficient
A=conv2(img, B, 'same');
%Filter for horizontal and vertical direction
KGx = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
KGy = [1, 2, 1; 0, 0, 0; -1, -2, -1];
%Convolution by image by horizontal and vertical filter
Filtered_X = conv2(A, KGx, 'same');
Filtered_Y = conv2(A, KGy, 'same');
%Calculate directions/orientations
arah = atan2 (Filtered_Y, Filtered_X);
arah = arah*180/pi;
pan=size(A,1);
leb=size(A,2);
%Adjustment for negative directions, making all directions positive
for i=1:pan
    for j=1:leb
        if (arah(i,j)<0)
            arah(i,j)=360+arah(i,j);
        end;
    end;
end;
arah2=zeros(pan, leb);
%Adjusting directions to nearest 0, 45, 90, or 135 degree
for i = 1  : pan
    for j = 1 : leb
        if ((arah(i, j) >= 0 ) && (arah(i, j) < 22.5) || (arah(i, j) >= 157.5) && (arah(i, j) <
202.5) || (arah(i, j) >= 337.5) && (arah(i, j) <= 360))
            arah2(i, j) = 0;
        elseif ((arah(i, j) >= 22.5) && (arah(i, j) < 67.5) || (arah(i, j) >= 202.5) && (arah(i,
j) < 247.5))
            arah2(i, j) = 45;
        elseif ((arah(i, j) >= 67.5 && arah(i, j) < 112.5) || (arah(i, j) >= 247.5 && arah(i, j) <
292.5))
            arah2(i, j) = 90;
        elseif ((arah(i, j) >= 112.5 && arah(i, j) < 157.5) || (arah(i, j) >= 292.5 && arah(i, j)
< 337.5))
            arah2(i, j) = 135;
        end;
    end;
end;
% figure, imagesc(arah2); colorbar;
%Calculate magnitude
magnitude = (Filtered_X.^2) + (Filtered_Y.^2);
magnitude2 = sqrt(magnitude);
BW = zeros (pan, leb);
```

```
%Non-Maximum Supression
for i=2:pan-1
    for j=2:leb-1
        if (arah2(i,j)==0)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i,j+1), magnitude2(i,j-1)]));
        elseif (arah2(i,j)==45)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j-1), magnitude2(i-1,j+1)]));
        elseif (arah2(i,j)==90)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j), magnitude2(i-1,j)]));
        elseif (arah2(i,j)==135)
            BW(i,j) = (magnitude2(i,j) == max([magnitude2(i,j),
magnitude2(i+1,j+1), magnitude2(i-1,j-1)]));
        end;
    end;
end;
BW = BW.*magnitude2;
% figure, imshow(BW);
%Hysteresis Thresholding
T_Low = T_Low * max(max(BW));
T_High = T_High * max(max(BW));
T_res = zeros (pan, leb);
for i = 1  : pan
    for j = 1 : leb
        if (BW(i, j) < T_Low)
            T_res(i, j) = 0;
        elseif (BW(i, j) > T_High)
            T_res(i, j) = 1;
        %Using 8-connected components
        elseif ( BW(i+1,j)>T_High || BW(i-1,j)>T_High || BW(i,j+1)>T_High ||
BW(i,j-1)>T_High || BW(i-1, j-1)>T_High || BW(i-1, j+1)>T_High || BW(i+1,
j+1)>T_High || BW(i+1, j-1)>T_High)
            T_res(i,j) = 1;
        end;
    end;
end;
edge_final = uint8(T_res.*255);

%Show final edge detection result
figure, imshow(edge_final);
```

метод NMS





Статистический метод

Формула подсчета среднего значения яркости в «рабочем окне»:

$$\mu = \frac{1}{m \cdot n} \cdot \sum_{i-1}^{m} \sum_{j-1}^{n} F(i, j)$$

Значение среднеквадратичного отклонения значений элементов рабочего окна от среднеарифметического значения:

$$\sigma = \sqrt{\frac{1}{m \cdot n} \cdot \sum_{i-1}^{m} \sum_{j-1}^{n} (F(i, j) - \mu)^2}$$

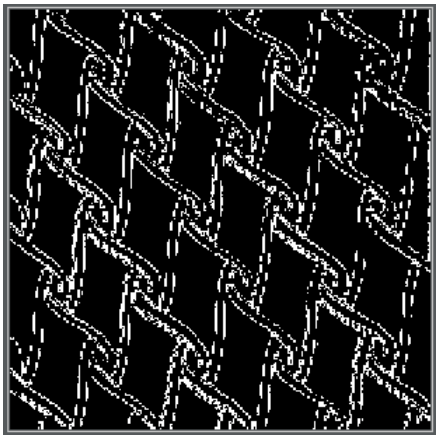
Модификация пикселей:

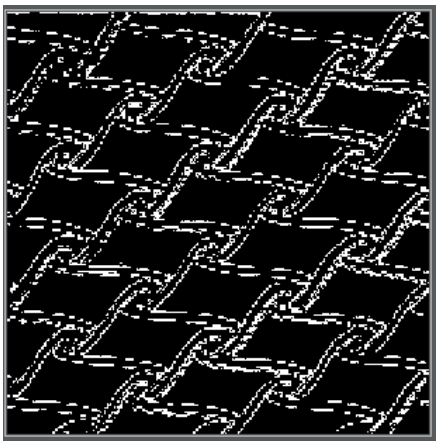
$$F'(i, j) = \sigma \cdot F(i, j)$$

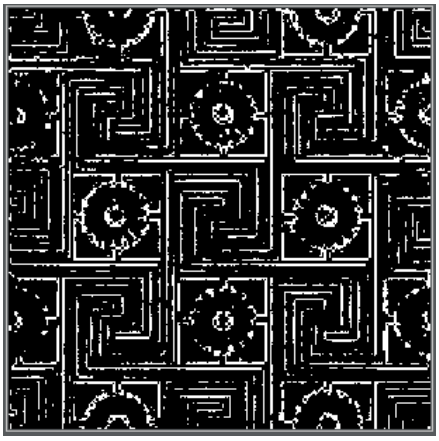
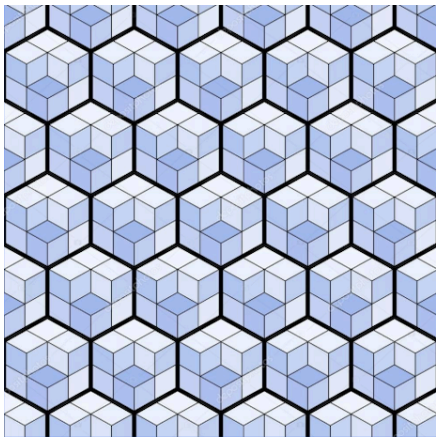
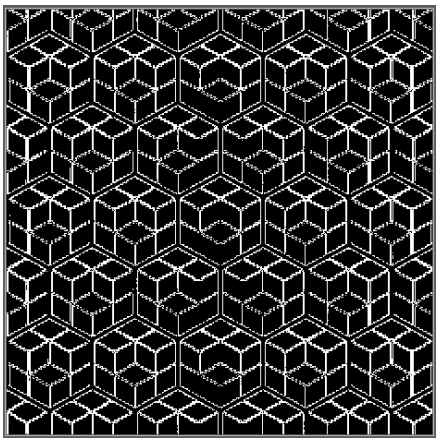
e. Статистический метод

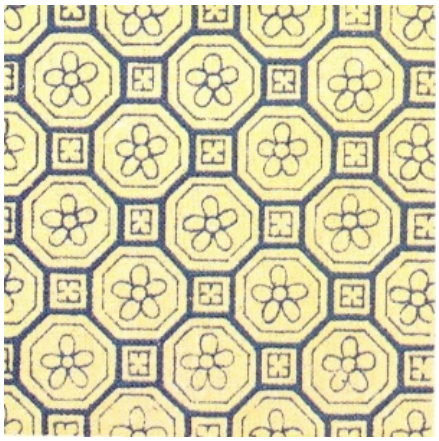
```
import math
import numpy as np
from skimage import io

FILENAME = 'regular_002.jpg'
WINDOW_SIZE = 2
THRESHOLD = 0.06
CONTOUR = 1
BACKGR = 0


def deviation(x: int, y: int, img: list) -> float:
    def avg_brightness(x: int, y: int, img: list) -> float:
        s = 0
        for i in range(x, x + WINDOW_SIZE):
            for j in range(y, y + WINDOW_SIZE):
                s += img[i][j]
        return s / (WINDOW_SIZE * WINDOW_SIZE)

    avg = avg_brightness(x, y, img)
    s = 0
    for i in range(x, x + WINDOW_SIZE):
        for j in range(y, y + WINDOW_SIZE):
            s += (img[i][j] - avg) ** 2

    return math.sqrt(s / (WINDOW_SIZE * WINDOW_SIZE))


def main():
    img = io.imread(FILENAME, as_gray=True)
    img = img.tolist()
    imglen = len(img)

    # создать таблицу сред яркостей в частях изобр размером с рабочее окно
    size = imglen // WINDOW_SIZE
    deviations = [[0] * size for _ in range(size)]

    print(f'computing brightness and deviation for img[{imglen}:{imglen}]')
    i = 0
    k = 0
    while i < imglen:
        j = 0
        m = 0
        while j < imglen:
            deviations[k][m] = deviation(i, j, img)
            m += 1
            j += WINDOW_SIZE
        i += WINDOW_SIZE
        k += 1

    print('adjusting input image')
    for i in range(imglen):
        for j in range(imglen):
            img[i][j] *= deviations[i // WINDOW_SIZE][j // WINDOW_SIZE]
            img[i][j] = CONTOUR if img[i][j] > THRESHOLD else BACKGR

    img_array = np.array(img)
    io.imsave('res.png', img_array)

if __name__ == '__main__':
    main()
```

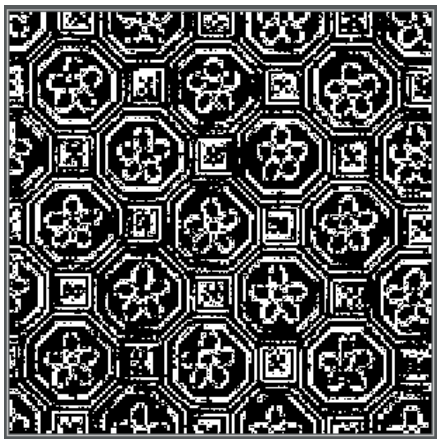
е. Статистический метод





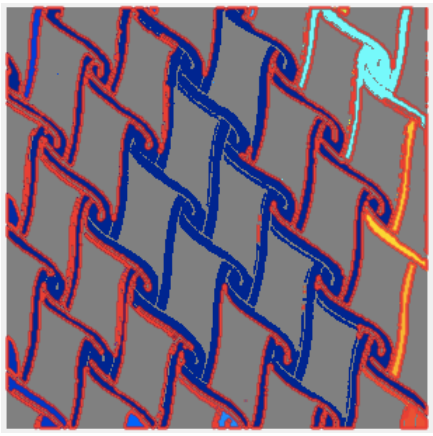


Задание 3

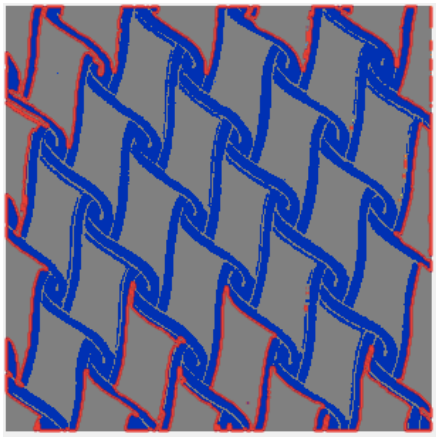
Граница должна быть выделена красным на рисунке.

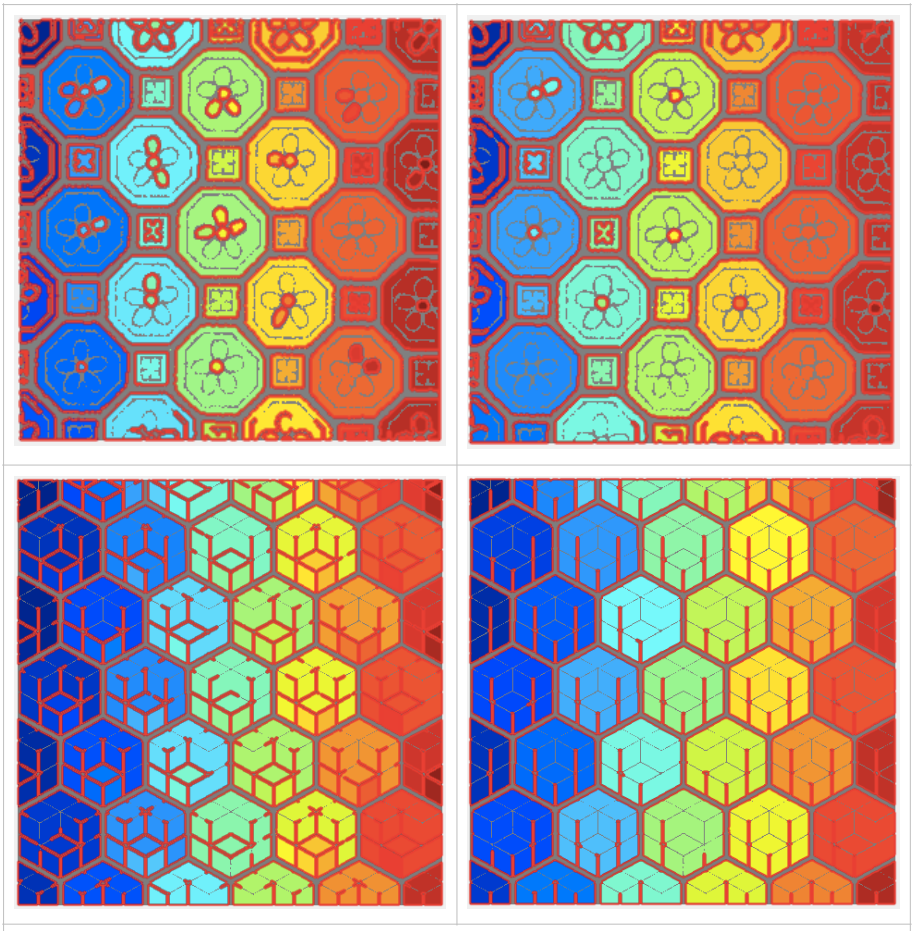
```
clear all;

% mark boundaries with red color
RED_COLOR = 'r';
COMPONENTS = 4; %8

img = imread('regular_002.jpg');
img = rgb2gray(img);
bin = imbinarize(img);

[B,L] = bwboundaries(bin, COMPONENTS, 'noholes');
imshow(label2rgb(L, @jet, [.5 .5 .5]))
hold on
for k = 1:length(B)
   boundary = B{k};
   x = boundary(:,2);
   y = boundary(:,1);
   plot(x, y, RED_COLOR, 'LineWidth', 2);
end
```

4-компонентное связывание



8-компонентное связывание



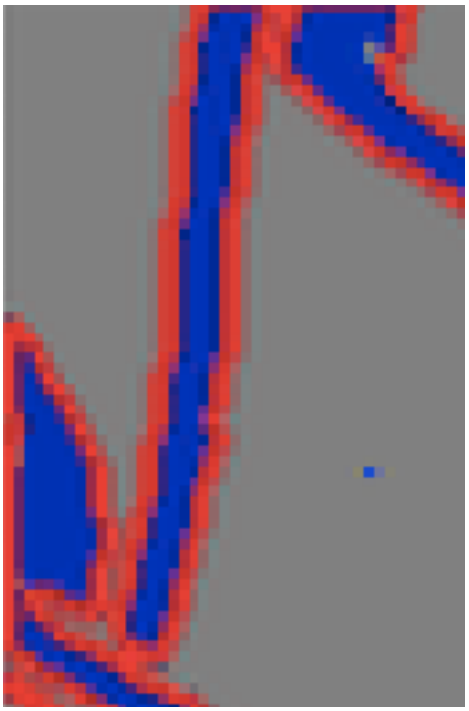


Можно заметить, что количество отдельных контуров тем больше, чем менее гибкий алгоритм выбора точек для контура. 4-х компонентное связывание может связывать только точку с ее верхним, нижним, левым и правым соседями. Таким образом под недодавшие в такое соединение точки создаются новые контуры

1. **Написать программу построения расхождений между выделенным и “реальным” контуром.**

Экспертный контур выделим **зеленым цветом** на рисунке. Ниже приведена его часть

Приведем аналогичную часть изображения, **красный контур** которого получен алгоритмическим путем



Изображения выше - часть регулярной текстуры на рисунке ниже. Т.к. тектура регулярная, дальнейшие результаты можно экстраполировать

Программа для сравнения и выдачи отклонений по заданному значению отклонения

```python
from skimage import io
import math
from prettytable import PrettyTable

# 1. размечаем в фотошопе границы текстуры в верхнем левом углу картинки
# 2. запускаем task3Machine.m -> сохраняем картинку

DEVIATION_MIN_DIST = 3.5
DEVIATION_MAX_DIST = 4.5

FILENAME_EXPERT = 'expert_31.png'
FILENAME_MACHINE = 'machine_31.png'

EXPERT_POINT_COLOR_THRESHOLD = [0, 210, 0]  # green
MACHINE_POINT_COLOR_THRESHOLD = [210, 0, 0]  # red

ROWS_TO_COMPARE = 30

RED = 0
GREEN = 1
BLUE = 2


class Point:
    def __init__(self, x: int, y: int):
        self.x = x
        self.y = y

    def euclidean_dist(self, another) -> float:
        return math.sqrt((self.x - another.x) ** 2 + (self.y - another.y) **
2)

    def __repr__(self):
        return self.__str__()

    def __str__(self):
        return f'[{self.x}:{self.y}]'


# отклонение
class Deviation:
    def __init__(self, expectation: Point, reality: Point):
        self.exp = expectation
        self.real = reality
        self.dist = expectation.euclidean_dist(reality)

    def __repr__(self):
        return self.__str__()

    def __str__(self):
        return f'Exp: {self.exp} Real: {self.real} dist: {self.dist}'


# цвет пикселя pixel[r,g,b] подходит под маску color[r,g,b]
# альфа-канал не учитываем
def is_of_color(color: list, pixel) -> bool:
    return pixel[RED] >= color[RED] and \
           pixel[GREEN] >= color[GREEN] and \
           pixel[BLUE] >= color[BLUE]
```

```python
def main():
    # полученное автоматическим путем изображение
    img_machine = io.imread(FILENAME_MACHINE)
    img_machine = img_machine.tolist()
    imglen_machine = len(img_machine)

    # экспертное изображение
    img_expert = io.imread(FILENAME_EXPERT)
    img_expert = img_expert.tolist()
    imglen_expert = len(img_expert)

    deviations = []
    i = 0
    while i < ROWS_TO_COMPARE:
        # ищем точку, поставленную экспертом
        expert_point = None
        for j in range(imglen_expert):
            if is_of_color(EXPERT_POINT_COLOR_THRESHOLD, img_expert[i][j]):
                expert_point = Point(i, j)
                break

        # ищем точку, поставленную алгоритмом
        machine_point = None
        for j in range(imglen_machine):
            if is_of_color(MACHINE_POINT_COLOR_THRESHOLD, img_machine[i][j]):
                machine_point = Point(i, j)
                break

        if expert_point and machine_point:
            deviations.append(Deviation(expert_point, machine_point))
            i += 1
        else:
            print(f'skipping row {i}. expert: {expert_point}, machine:
{machine_point}')
            i += 1

    [print(d) for d in deviations]

    # находим отклоенения попадающие под THRESHOLD
    deviations_thr = list(filter(lambda d: DEVIATION_MIN_DIST < d.dist <
DEVIATION_MAX_DIST, deviations))

    print(f'\npoints with deviation in range: ({DEVIATION_MIN_DIST},
{DEVIATION_MAX_DIST})')
    [print(d) for d in deviations]

    # печать таблицы
    t = PrettyTable(['Ожидание', 'Реальность', 'Расхождение'])
    for d in deviations:
        t.add_row([d.exp, d.real, d.dist])

    print(t)

    print(f'Percentage with deviation: {len(deviations_thr)/len(deviations)}')


if __name__ == '__main__':
    main()
```

Пример части таблицы отклонений

```
+----------+------------+-------------+
| Ожидание | Реальность | Расхождение |
+----------+------------+-------------+
| [0:288]  |  [0:283]   |     5.0     |
| [1:287]  |  [1:283]   |     4.0     |
| [2:287]  |  [2:283]   |     4.0     |
| [3:287]  |  [3:283]   |     4.0     |
| [4:287]  |  [4:282]   |     5.0     |
| [5:287]  |  [5:282]   |     5.0     |
| [6:287]  |  [6:282]   |     5.0     |
| [7:287]  |  [7:282]   |     5.0     |
| [8:287]  |  [8:282]   |     5.0     |
| [9:286]  |  [9:281]   |     5.0     |
| [10:285] |  [10:281]  |     4.0     |
| [11:285] |  [11:281]  |     4.0     |
| [12:285] |  [12:281]  |     4.0     |
```

Пример части таблицы отклонений для отклонений в промежутке от 3.5 до 4.5

```
+----------+------------+-------------+
| Ожидание | Реальность | Расхождение |
+----------+------------+-------------+
| [1:287]  |  [1:283]   |     4.0     |
| [2:287]  |  [2:283]   |     4.0     |
| [3:287]  |  [3:283]   |     4.0     |
| [10:285] |  [10:281]  |     4.0     |
| [11:285] |  [11:281]  |     4.0     |
| [12:285] |  [12:281]  |     4.0     |
| [14:285] |  [14:281]  |     4.0     |
| [15:285] |  [15:281]  |     4.0     |
| [19:284] |  [19:280]  |     4.0     |
| [21:284] |  [21:280]  |     4.0     |
| [24:284] |  [24:280]  |     4.0     |
| [25:284] |  [25:280]  |     4.0     |
| [28:283] |  [28:279]  |     4.0     |

Percentage with deviation:
0.43333333333333335
```

2. Написать программу которая по таблице определяла процент точек с заданным отклонением.

Программа выше строит таблицу отклонений и позволяет определить процент точек с отклонением (43%)

Задание 4

Граница должна быть представлена в следующих видах

- Кодирование по трем признакам
- Кодирование трех разрядным кодом
- Кодирование проекциями
- Кодирование Координатами концов векторов
- Кодирование восемью комплексными числами
- Кодирование в полярных координатах

Программа, реализующая кодирование по всем приведенным выше признакам

```
import math
from enum import Enum
import matplotlib.pyplot as plt
from skimage import io

WHITE_CONTOUR_MASK = [200, 200, 200]
RED = 0
GREEN = 1
BLUE = 2

# FILENAME = 'circle.png'
# FILENAME = 'star.png'
FILENAME = 'pine5.jpg'
IMG = io.imread(FILENAME).tolist()

STEP = 29
SAMPLES = 7


class Direction(Enum):
    CLOCKWISE = 1
    COUNTER_CLOCKWISE = -1


class Point:
    def __init__(self, x: int, y: int):
        self.x = x
        self.y = y
        try:
            self.is_contour = is_contour(IMG[self.x][self.y])
        except IndexError:
            pass

    def __eq__(self, other):
        return other is not None and \
               self.x == other.x and self.y == other.y

    def __repr__(self):
        return self.__str__()

    def __str__(self):
        return f'[{self.x}:{self.y}] c:{self.is_contour}'


class Vector:
    def __init__(self, pt_from: Point, pt_to: Point):
        self.pt_from = pt_from
        self.pt_to = pt_to

    # радиус-вектор представляется точкой
    def to_radius_vector(self) -> Point:
        delta_x = self.pt_to.x - self.pt_from.x
        delta_y = self.pt_to.y - self.pt_from.y
        return Point(delta_x, delta_y)

    def __repr__(self):
        return self.__str__()

    def __str__(self):
        return f'{self.pt_from} -> {self.pt_to}'
```

```python
class Encoding:
    def encode(self) -> str:
        raise NotImplementedError



class PolarCoordinatesEncoding(Encoding):
    """
    Кодирование в полярных координатах
    """

    def __init__(self, v_curr: Vector):
        self.v_curr = v_curr
        self.angle = self._polar_angle(v_curr)
        self.len = vector_len(v_curr)

    def _polar_angle(self, v: Vector) -> float:
        ox_vector = Vector(Point(0, 0), Point(1, 0))
        radius_vec = Vector(Point(0, 0), v.pt_to)
        return directionwise_angle(radius_vec, ox_vector)

    def encode(self) -> str:
        return f'{self.len:.1f} + cos({self.angle:.1f})'



class ThreeAttrEncoding(PolarCoordinatesEncoding):
    """
    Элемент границы, закодированной по трем признакам:
    (длина вектора, угол, направление поворота к следующему вектору)
    """

    def __init__(self, v_curr: Vector, v_next: Vector):
        super(ThreeAttrEncoding, self).__init__(v_curr)
        angle = self._normalize_angle(v_curr, v_next)
        self.direction = Direction.COUNTER_CLOCKWISE if angle > 0 else
Direction.CLOCKWISE
        self.angle = abs(angle)

    # угол [0 360) преобразовать в угол [0 180) со знаком
    # например, 300" == -60"
    def _normalize_angle(self, v1: Vector, v2: Vector) -> float:
        big_angle = directionwise_angle(v1, v2)
        return big_angle if big_angle <= 180 else big_angle - 360

    def encode(self) -> str:
        return f'[{self.len:.1f} \tangle:{self.angle:.1f}
{self.direction}]'
```

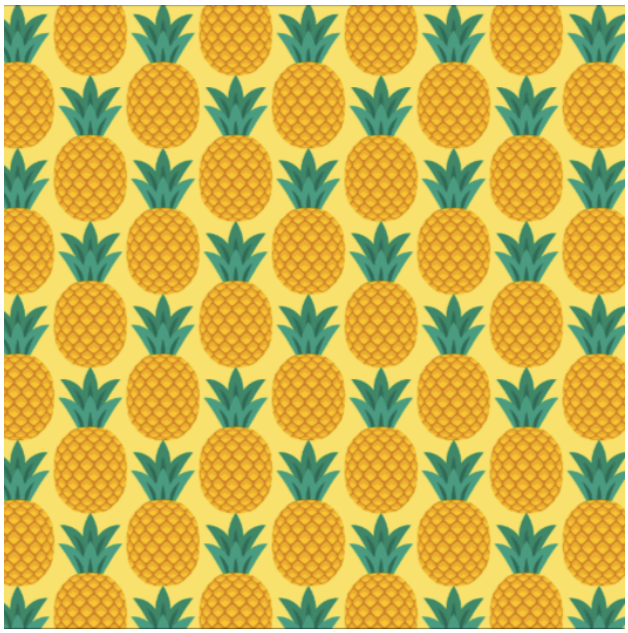
```python
class ThreeDigitEncoding(Encoding):
    """
    трехразрядный код
    """

    def __init__(self, v_curr: Vector):
        self.v = v_curr
        ox_vector = Vector(Point(0, 0), Point(1, 0))
        self.code = self._three_digit_code(v_curr, ox_vector)

    def _three_digit_code(self, v1: Vector, v2: Vector) -> int:
        big_angle = directionwise_angle(v2, v1)
        code = None
        if 337.5 <= big_angle < 360:
            code = 0
        elif 292.5 <= big_angle < 337.5:
            code = 1
        elif 247.5 <= big_angle < 292.5:
            code = 2
        elif 202.5 <= big_angle < 247.5:
            code = 3
        elif 157.5 <= big_angle < 202.5:
            code = 4
        elif 112.5 <= big_angle < 157.5:
            code = 5
        elif 67.5 <= big_angle < 112.5:
            code = 6
        elif 22.5 <= big_angle < 67.5:
            code = 7
        elif big_angle < 22.5:
            code = 0
        # в бинарный вид
        return code

    def encode(self) -> str:
        return f'{self.code}: {self.code:03b}'



class ProjectionEncoding(ThreeDigitEncoding):
    """
    кодирование проекциями
    """

    def __init__(self, v_curr: Vector):
        super(ProjectionEncoding, self).__init__(v_curr)
        self.projection = self._project(self.code)

    def _project(self, code: int) -> (int, int):
        code_to_projection = {
            0: (0, 0),
            1: (1, -1),
            2: (0, -1),
            3: (-1, -1),
            4: (-1, 0),
            5: (-1, 1),
            6: (0, 1),
            7: (1, 1)
        }
        return code_to_projection[code]

    def encode(self) -> str:
        return f'{self.projection}'
```

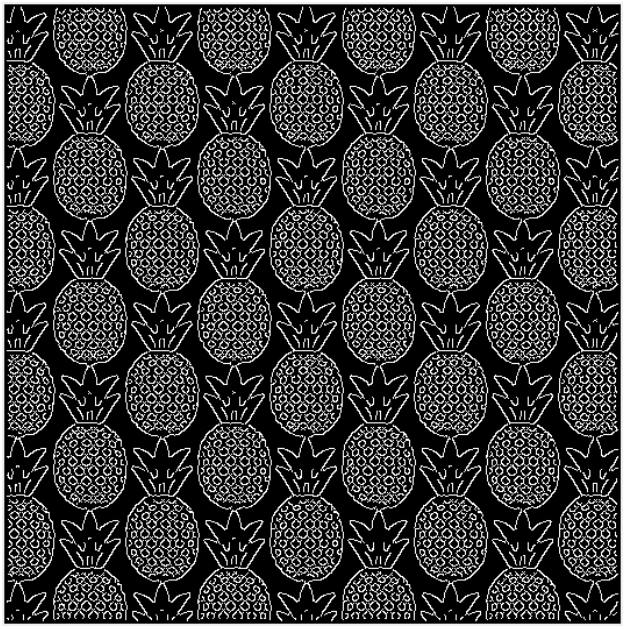
```
class ComplexNumberEncoding(ProjectionEncoding):
    """
    Кодирование восемью комплексными числами
    """

    def __init__(self, v_curr: Vector):
        super(ComplexNumberEncoding, self).__init__(v_curr)

    def encode(self) -> str:
        real = self.projection[0]
        imag = self.projection[1]
        if 0 == real and 0 == imag:
            return '0'
        elif 0 == real:
            return f'({imag}i)'
        elif 0 == imag:
            return f'({real})'
        else:
            return f'({real} + {imag}i)'



class VectorCoordinatesEncoding(ThreeDigitEncoding):
    """
    Кодирование Координатами концов векторов
    """

    def __init__(self, v_curr: Vector):
        super(VectorCoordinatesEncoding, self).__init__(v_curr)

    def encode(self):
        return f'{self.v.pt_from} -> {self.v.pt_to}'




# угол между радиус-векторами в диапазоне [0, 360)
def directionwise_angle(v1: Vector, v2: Vector) -> float:
    p1, p2 = v1.to_radius_vector(), v2.to_radius_vector()

    v1_theta = math.atan2(p1.y, p1.x)
    v2_theta = math.atan2(p2.y, p2.x)
    r = (v2_theta - v1_theta) * (180.0 / math.pi)
    if r < 0:
        r += 360.0
    return r


def vector_len(v: Vector) -> float:
    return math.sqrt((v.pt_to.x - v.pt_from.x) ** 2 + (v.pt_to.y -
v.pt_from.y) ** 2)


# pixel = [R, G, B]
def is_contour(pixel) -> bool:
    def is_of_color(color: list) -> bool:
        return pixel[RED] >= color[RED] and \
               pixel[GREEN] >= color[GREEN] and \
               pixel[BLUE] >= color[BLUE]

    return is_of_color(WHITE_CONTOUR_MASK)
```

```python
def extract_contour_pts(img) -> list:
    rows = len(img)
    cols = len(img[0])

    # находим первую попавшуюся точку контура
    def find_start_pt() -> Point:
        for i in range(rows):
            for j in range(cols):
                if is_contour(img[i][j]):
                    return Point(i, j)

    start_pt = find_start_pt()
    curr_pt = start_pt
    contour = []
    while True:
        contour.append(curr_pt)
        x, y = curr_pt.x, curr_pt.y

        nearest_points = [
            Point(x + 1, y),
            Point(x + 1, y - 1),
            Point(x, y - 1),
            Point(x - 1, y - 1),
            Point(x - 1, y),
            Point(x - 1, y + 1),
            Point(x, y + 1),
            Point(x + 1, y + 1)
        ]
        next_pt = None
        # рассматриваем 8 соседей
        # среди соседей будут несколько контурных точек
        for p in nearest_points:
            # нам надо найти еще НЕ посещенную точку
            if p.is_contour and p not in contour:
                next_pt = p
                break
        # не нашли непосещенной точки => выходим
        if next_pt is None:
            break

        curr_pt = next_pt

    print(f'contour size: {len(contour)}')
    print(f'start pt: {start_pt}')
    print(f'end   pt: {contour[len(contour) - 1]}')

    # draw image normally
    for p in contour:
        p.x, p.y = p.y, rows - p.x
    return contour



def approximate_contour(contour: list, step) -> list:
    vectors = []
    ctr_len = len(contour)
    for i in range(0, ctr_len - step, step):
        p0 = contour[i]
        p1 = contour[i + step]
        vectors.append(Vector(p0, p1))
    # замкнуть контур
    first = vectors[0]
    last = vectors[-1]
    vectors.append(Vector(last.pt_to, first.pt_from))
    return vectors
```

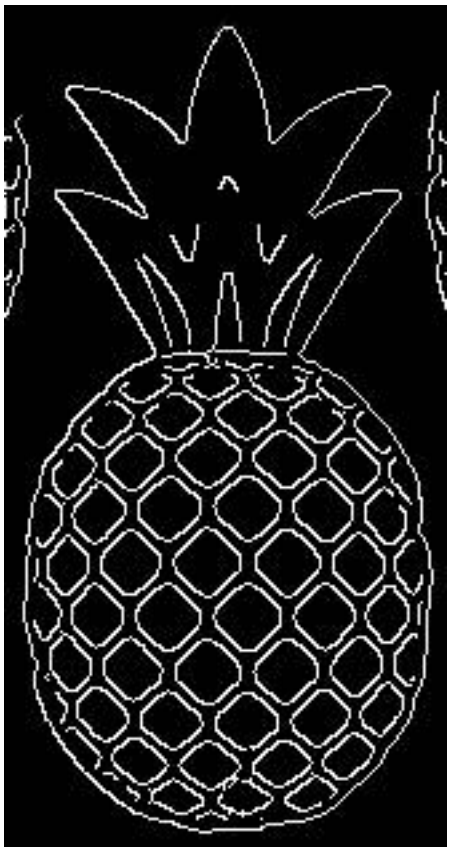
```
def main():
    points = extract_contour_pts(IMG)
    vectors = approximate_contour(points, STEP)
    vec_len = len(vectors)

    print('\nКодирование по трем признакам')
    enc1 = []
    for i in range(vec_len):
        v1 = vectors[i]
        v2 = vectors[(i + 1) % vec_len]
        enc1.append(ThreeAttrEncoding(v1, v2))
    [print(e.encode()) for e in enc1[:SAMPLES]]

    print('\nКодирование трехразрядным кодом')
    enc2 = []
    for i in range(vec_len):
        enc2.append(ThreeDigitEncoding(vectors[i]))
    [print(e.encode()) for e in enc2[:SAMPLES]]

    print('\nКодирование проекциями')
    enc3 = []
    for i in range(vec_len):
        enc3.append(ProjectionEncoding(vectors[i]))
    [print(e.encode()) for e in enc3[:SAMPLES]]

    print('\nКодирование восемью комплексными числами')
    enc4 = []
    for i in range(vec_len):
        enc4.append(ComplexNumberEncoding(vectors[i]))
    [print(e.encode()) for e in enc4[:SAMPLES]]

    print('\nКодирование координатами концов векторов')
    enc5 = []
    for i in range(vec_len):
        enc5.append(VectorCoordinatesEncoding(vectors[i]))
    [print(e.encode()) for e in enc5[:SAMPLES]]

    print('\nКодирование в полярных координатах')
    enc6 = []
    for i in range(vec_len):
        enc6.append(PolarCoordinatesEncoding(vectors[i]))
    [print(e.encode()) for e in enc6[:SAMPLES]]

    for v in vectors:
        plt.plot([v.pt_from.x, v.pt_to.x], [v.pt_from.y, v.pt_to.y])
    plt.show()


if __name__ == '__main__':
    main()
```

Пример работы
Возьмем изображение следующей текстуры и выделим на нем контуры наилучшим способом (например, NMS)

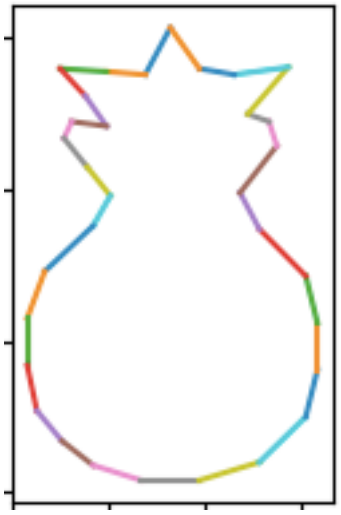




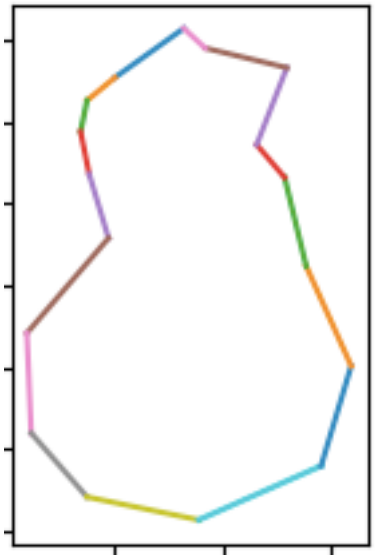
Выделим из изображения один конкретный образ (можем этот сделать т.к. Текстура регулярная)



Аппроксимируем кривые линии полученного контура (только внешнего) с шагом в, например, 30 пикселей и выведем получившийся результат:



N = 30



N=60

После экстраполяции становится видны получившиеся вектора.

Стартуем с самой верхней части ананаса, уходя в обход против часовой стрелки (налево)

Результаты кодирования получившейся границы
(для наглядности показаны только 7 первых элементов)

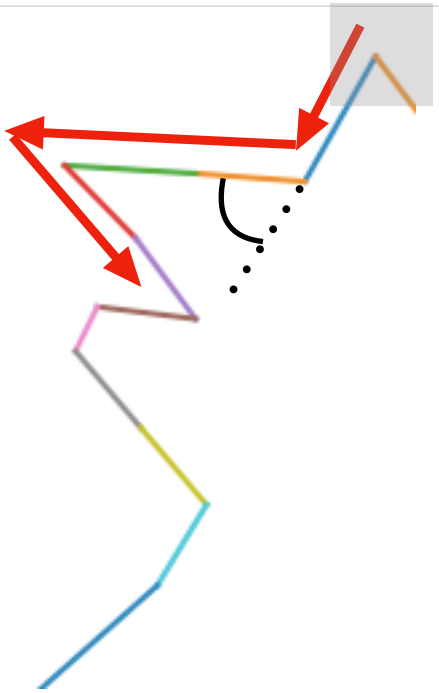
Кодирование по трем признакам

```
[33.6       angle:73.0 Direction.CLOCKWISE]
[20.1       angle:0.9 Direction.COUNTER_CLOCKWISE]
[24.1       angle:130.6 Direction.COUNTER_CLOCKWISE]
[22.2       angle:7.0 Direction.CLOCKWISE]
[22.8       angle:128.3 Direction.CLOCKWISE]
[18.2       angle:79.5 Direction.COUNTER_CLOCKWISE]
[11.7       angle:52.3 Direction.COUNTER_CLOCKWISE]
```

Кодирование трехразрядным кодом

```
3: 011
4: 100
4: 100
1: 001
1: 001
4: 100
2: 010
```

На рисунке справа можно проверить корректность кодирования границ, мысленно нарисовав квадрат с 8 соседями или же проверив повороты по/против часовой



Кодирование проекциями

```
(-1, -1)
(-1, 0)
(-1, 0)
(1, -1)
(1, -1)
(-1, 0)
(0, -1)
```

Кодирование восемью комплексными числами

```
(-1 + -1i)
(-1)
(-1)
(1 + -1i)
(1 + -1i)
(-1)
(-1i)
```

Кодирование координатами концов векторов

```
[82:307] c:True -> [69:276] c:True
[69:276] c:True -> [49:278] c:True
[49:278] c:True -> [25:280] c:True
[25:280] c:True -> [38:262] c:True
[38:262] c:True -> [49:242] c:True
[49:242] c:True -> [31:245] c:True
[31:245] c:True -> [27:234] c:True
```

Кодирование в полярных координатах

```
33.6 + cos(284.0)
20.1 + cos(280.0)
24.1 + cos(275.1)
22.2 + cos(278.3)
22.8 + cos(281.4)
18.2 + cos(277.2)
11.7 + cos(276.6)
```

Задание 5

Наилучшим методом выделения контура на моей тестовой выборке является метод локальных максимумов, т.к. он работает наиболее обощенного и справляется в большинстве случаев хорошо.

Метод порогового значения, например, надо настраивать под конкретное изображение для достижения наилучших результатов.

Статистический метод также требует настройки окна и некоторой постобработки. Скелетизация работает тоже достаточно хорошо в среднем случае.

Однако, пороговый метод и метод дифференцирования крайне просты в реализации (по сравнению с NMS и статистическим методом) и это их значительный плюс.

Таким образом, при выборе метода необходимо соблюдать баланс между качеством работы и простотой алгоритма.