МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский университет
«Московский институт электронной техники»

Отчёт по результатам предпроектного командного обучения по дисциплине «Основы языка Java»

«Шифр Мандельброта»

Подготовили
студенты группы ПИН-36:
Бойков И.И.
Котляров Д.Н.
Хаснаш А.

Москва, 2024

Анализ применимости множества Мандельброта для шифрования изображений

Множество Мандельброта - множество точек c на комплексной плоскости, для которых рекуррентное соотношение $z_{n+1} = z_n^2 + c$ при $z_0 = 0$ задаёт ограниченную последовательность.

Для построения множества Мандельброта на Java обозначим действительную ось, как ось x, мнимую ось – как ось y, установим максимально допустимое число итераций, за которые принимается решение, принадлежит точка множеству, или нет, равным *MAX_ITER* (пусть 3000). Тогда множество Мандельброта можно построить, используя следующий код:

```
for (int y = 0; y < getHeight; y++) {
    double zx = 0, zy = 0;
    double cX = (x - getWidth / 1.75) / ZOOM;
    double cY = (y - getHeight / 1.75) / ZOOM;
    int i = MAX_ITER;
    while (zx * zx + zy * zy < 4 && i > 0) {
        double tmp = zx * zx - zy * zy + cX;
        zy = 2.0 * zx * zy + cY;
        zx = tmp;
        i--;
    }
    int color = i | (i << 10) | (i << 14);
    image.setRGB(x, y, i > 0 ? color : 0);
}
```

Листинг 1. Итерационный процесс построения множества.

Получим графическое изображение множества Мандельброта:

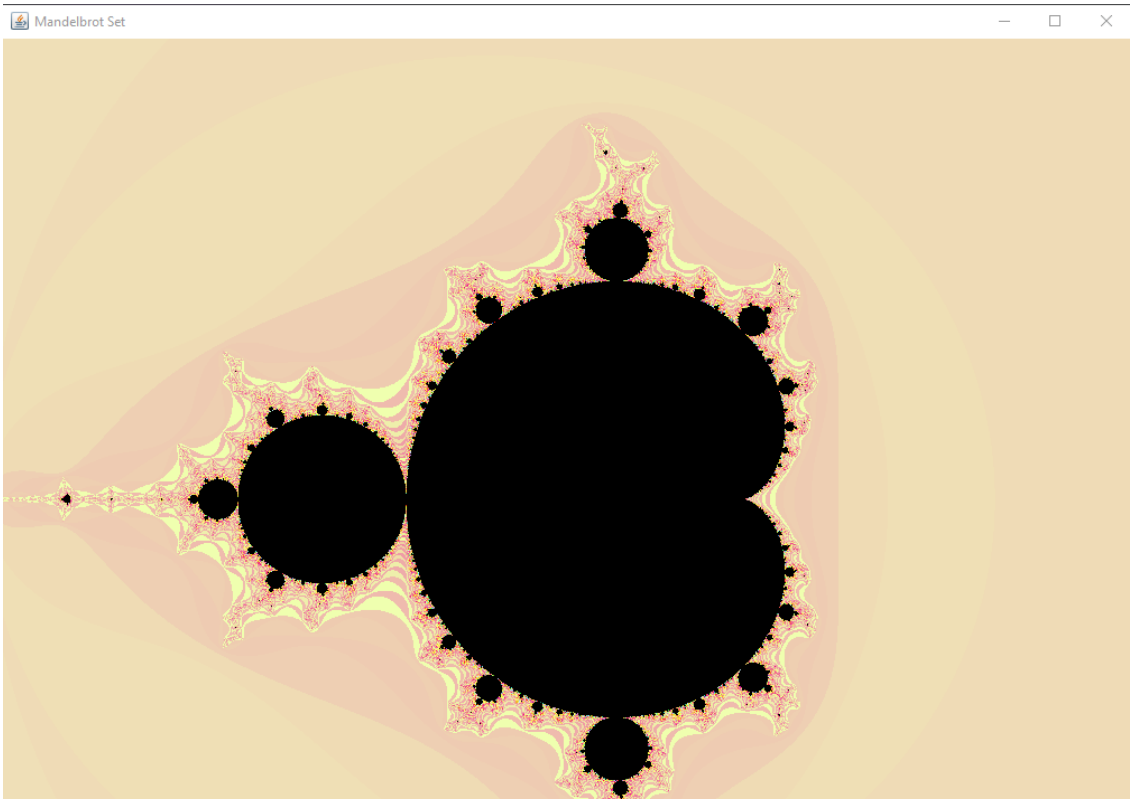


Рисунок 1. Графическое изображение множества Мандельброта

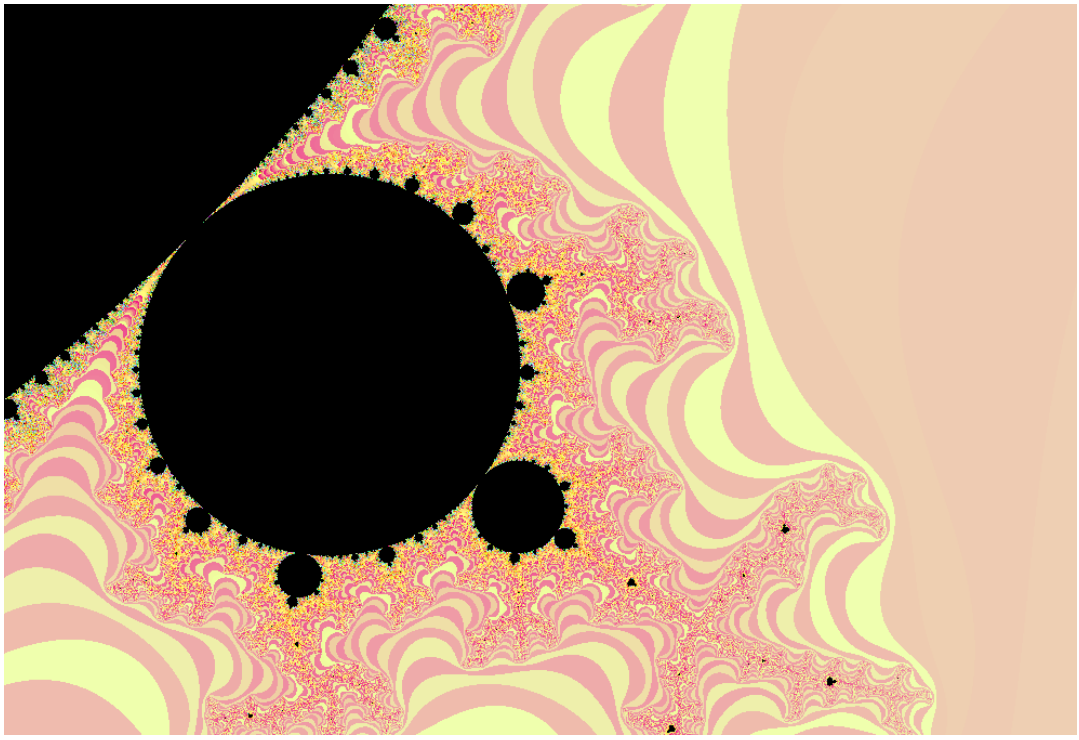


Рисунок 2. Граница множества Мандельброта при 13-кратном увеличении

Наблюдая за множеством, нетрудно видеть, что оно (в особенности - его граница) представляет собой сложную структуру нетривиальной формы, состоящую из множества объектов различных цветов, что позволяет предположить, что шифрование и дешифрование сообщений можно осуществлять путём смешивания шифруемого изображения с изображением множества Мандельброта с помощью операции XOR.

В качестве шифрующей функции будем рассматривать отображение $M: R^4 \to R^{n \cdot m}, M(x_1, x_2, x_3, x_4) = A$, где x_1, x_2, x_3, x_4 - параметры ключа, A – матрица размерности n на m пикселей. Чтобы функция M считалась применимой для шифрования, наложим на неё два требования:

1. Односторонность: эмпирическая вероятность P правильного обращения функции M (определения параметров x_1, x_2, x_3, x_4 по значениям элементов матрицы A) была менее 0,01.
2. Чувствительность ключа к изменениям $k = 1 - S$ должна быть не менее 0.99, где S – показатель сходства различных изображений-ключей.

Эмпирическую вероятность P правильного обращения функции M будем определять путём сравнения сгенерированных изображений-ключей с параметрами шифрования x_1, x_2, x_3, x_4 с изображением-ключом с известными параметрами шифрования C_1, C_2, C_3, C_4. В случае, если показатель сходства между изображениями S составит 0,9 или более, положим предсказанные параметры ${x_1}^{(1)} = C_1, {x_2}^{(1)} = C_2, {x_3}^{(1)} = C_3, {x_4}^{(1)} = C_4$, иначе предсказанные параметры будем считать неопределёнными. Далее, будем сравнивать предсказанные параметры с истинными, и в случае их совпадения вероятность правильного предсказания i-го параметра при j-м сравнении $p_{ij} = 1$. Здесь $i \in [1; 4], j \in [1; n]$, n – число сравнений. Эмпирическую вероятность P будем определять по формуле:

$$P = 0{,}25\left(\frac{\sum_{j=1}^{n} p_{1j}}{n} + \frac{\sum_{j=1}^{n} p_{2j}}{n} + \frac{\sum_{j=1}^{n} p_{3j}}{n} + \frac{\sum_{j=1}^{n} p_{4j}}{n}\right)$$

Показатель сходства между изображениями S будем определять тремя независимыми способами:

1. Процент совпадения значений пикселей α, определяемый путём попарного сравнения изображений (значений элементов матриц A_1, A_2 размерности n на m).
2. Индекс структурного сходства между изображениями $\theta(A_1, A_2)$:
$$\theta(A_1, A_2) = \frac{(2\mu_1\mu_2 + C_1)(2\sigma_{12} + C_2)}{(\mu_1^2+\mu_2^2 + C_1)(\sigma_1^2 + \sigma_2^2 + C_2)}$$
Где A_1, A_2 - сравниваемые изображения, μ_1, μ_2 - средние значения яркости изображений, σ_1^2, σ_2^2 - её дисперсии, $2\sigma_{12}$ – ковариация яркостей изображений.
3. Коэффициент корреляции Пирсона ρ между пикселями изображений:
$$\rho = \frac{\sum_{i,j=1}^{n,m}\left(a_{1_{ij}} - \overline{a_1}\right)(a_{2_{ij}} - \overline{a_2})}{\sqrt{\sum_{i,j=1}^{n,m}(a_{1_{ij}} - \overline{a_1})^2 \sum_{i,j=1}^{n,m}(a_{2_{ij}} - \overline{a_2})^2}}$$
Где $a_{1_{ij}}, a_{2_{ij}}$ - значения яркости пикселей изображений, $\overline{a_1}, \overline{a_2}$ - средние значения яркости изображений.

Для проверки односторонности преобразования и чувствительности изображения-ключа к изменениям параметров рассмотрим 6 различных наборов параметров. Параметры, не указанные в списке, будут зафиксированы, оценка правильности их предсказания проводиться не будет.

1. $x_1 = Re(Z_0) \sim R[-5; 5]$ - равномерно распределённая случайная величина;
$x_2 = Im(Z_0) \sim R[-5; 5]$ - равномерно распределённая случайная величина;
$x_3 = Re(c) \sim R[-512; 512]$ - равномерно распределённая случайная величина;
$x_4 = Im(c) \sim [-384; 384]$ - равномерно распределённая случайная величина;

2. $x_1 = Re(Z_0) \sim R[-5; 5]$ - равномерно распределённая случайная величина;
$x_2 = Im(Z_0) \sim R[-5; 5]$ – равномерно распределённая случайная величина;

3. $x_1 = Re(Z_0) \sim R[-5; 5]$ - равномерно распределённая случайная величина;

4. $x_2 = Im(Z_0) \sim R[-5; 5]$ - равномерно распределённая случайная величина;

5. $x_1 = N \sim R[800; 1700]$ - равномерно распределённая случайная величина;
$x_2 = Z \sim R[100; 144] \cdot 1000$ - равномерно распределённая случайная величина;
$x_3 = \Delta x \sim R[-0{,}9998; 0{,}9998]$ – равномерно распределённая случайная величина;
$x_4 = \Delta y \sim [-0{,}9998; 0{,}9998]$ – равномерно распределённая случайная величина;

6. Аналогично п. 5, дополнительно наложим требование: более 500 уникальных значений цветов $a_{ij} \in A$, отношение количества пикселей одинакового цвета к общему количеству пикселей в изображении $\frac{c_{ij}}{a_{ij}} < 0{,}25$.

Реализация этого условия на Java:

```
private boolean checkImageDiversity(BufferedImage image) {
        Map<Integer, Integer> colorCount = new HashMap<>();
        int totalPixels = image.getWidth() * image.getHeight();

        for (int x = 0; x < image.getWidth(); x++) {
            for (int y = 0; y < image.getHeight(); y++) {
                int color = image.getRGB(x, y);
                colorCount.put(color, colorCount.getOrDefault(color, 0) +
1);
            }
        }

        int uniqueColors = colorCount.size();
        int maxCount =
colorCount.values().stream().max(Integer::compare).orElse(0);
        double percentage = (double) maxCount / totalPixels;

        return (uniqueColors > 500 && percentage < 0.25);
    }
```

Листинг 2. Проверка количества цветов и площади, занимаемой ими, на сгенерированном изображении.

В качестве известных параметров ключа C_1, C_2, C_3, C_4 выберем нули, при которых имеем стандартное изображение множества (Рисунок 1).

Для определения односторонности преобразования по вышеописанной методике будем проводить n = 1000 сравнений изображений разрешения 1024x720 пикселей.

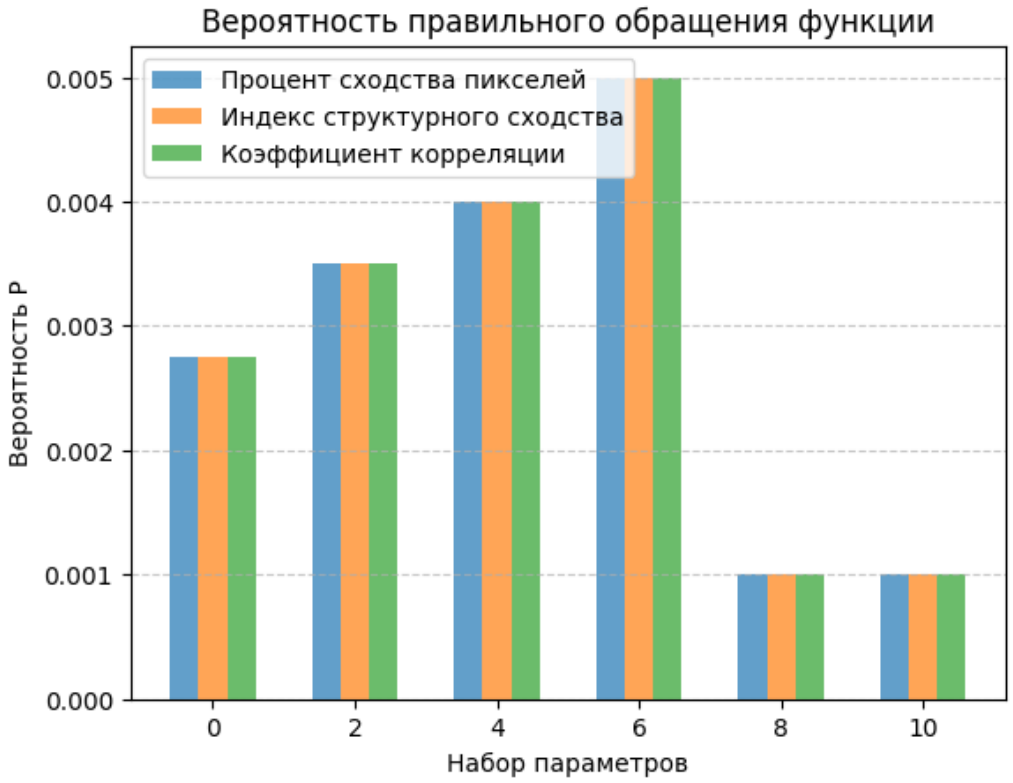


Рисунок 3. Зависимость вероятности предсказания параметров от выбора параметров

Такое же количество n сравнений тех же изображений-ключей, сгенерированных по тем же параметрам, проведём для определения чувствительности изображений-ключей к изменениям параметров k:

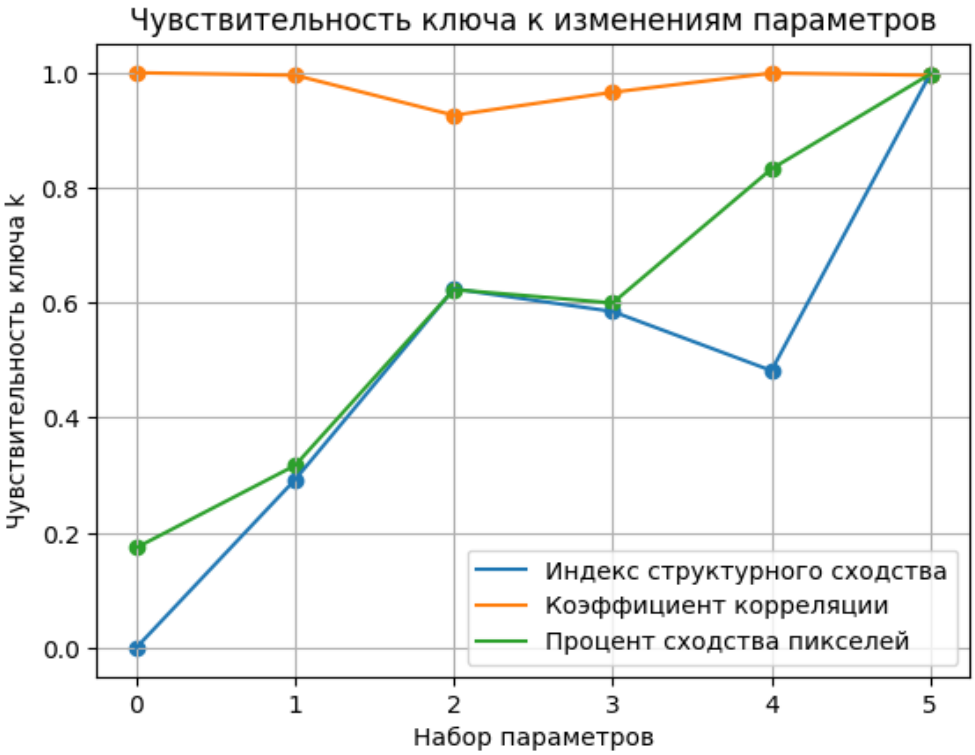


Рисунок 4. Зависимость чувствительности ключа к изменениям параметров от выбора параметров

Анализ результатов сравнений показывает, что функция M применима для шифрования изображений только при выборе набора параметров 6: число итераций, за которое принимается решение, принадлежит ли точка множеству Мандельброта N, увеличение изображения множества Z, сдвиги по горизонтальной и вертикальной осям $\Delta x, \Delta y$ соответственно. Также должна проверяться цветность изображения G и площадь одноцветных участков L.

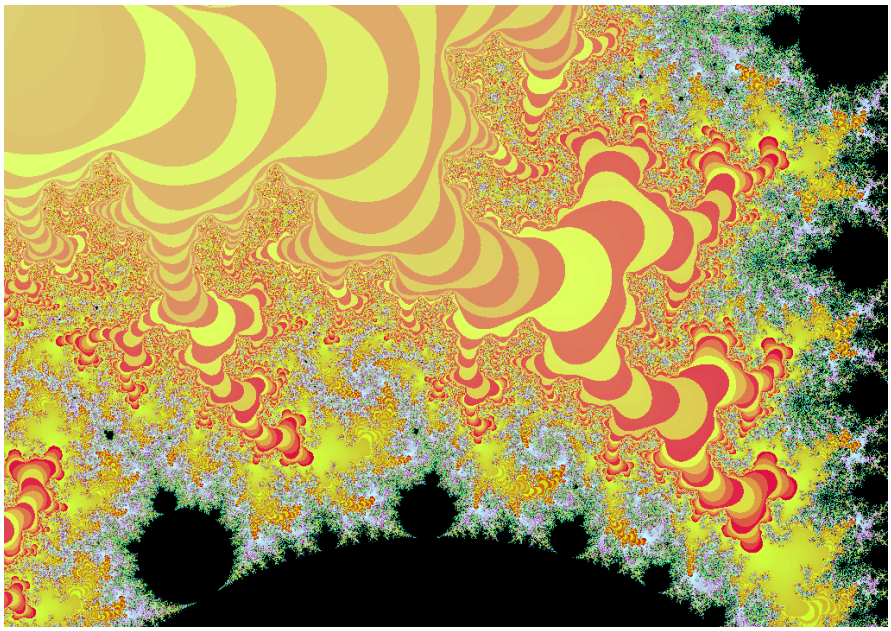


Рисунок 5. Пример изображения-ключа при выборе параметров №6

Алгоритм шифрования и дешифрования изображений

Предлагаемый алгоритм шифрования строится на коммутативности операции XOR, или сложения по модулю 2. Представим изображения A (шифруемое) и B (шифрующее) в виде матриц n x m, состоящих из целочисленных элементов, содержащих значения цветов пикселей. Пусть C (зашифрованное изображение) = $A \oplus B$, где $A \oplus B = a_{ij}\ XOR\ b_{ij}, i \in [1; n], j \in [1; m]$. Докажем, что если $A \oplus B = C$, то $C \oplus B = A$.

Пусть $A \oplus B = C$. Это означает, что для каждого элемента матриц выполняется $a_{ij} \oplus b_{ij} = c_{ij}$. Представим уравнение в виде $(A \oplus B) \oplus B = C \oplus B$. Это означает, что для каждого элемента матриц выполняется:

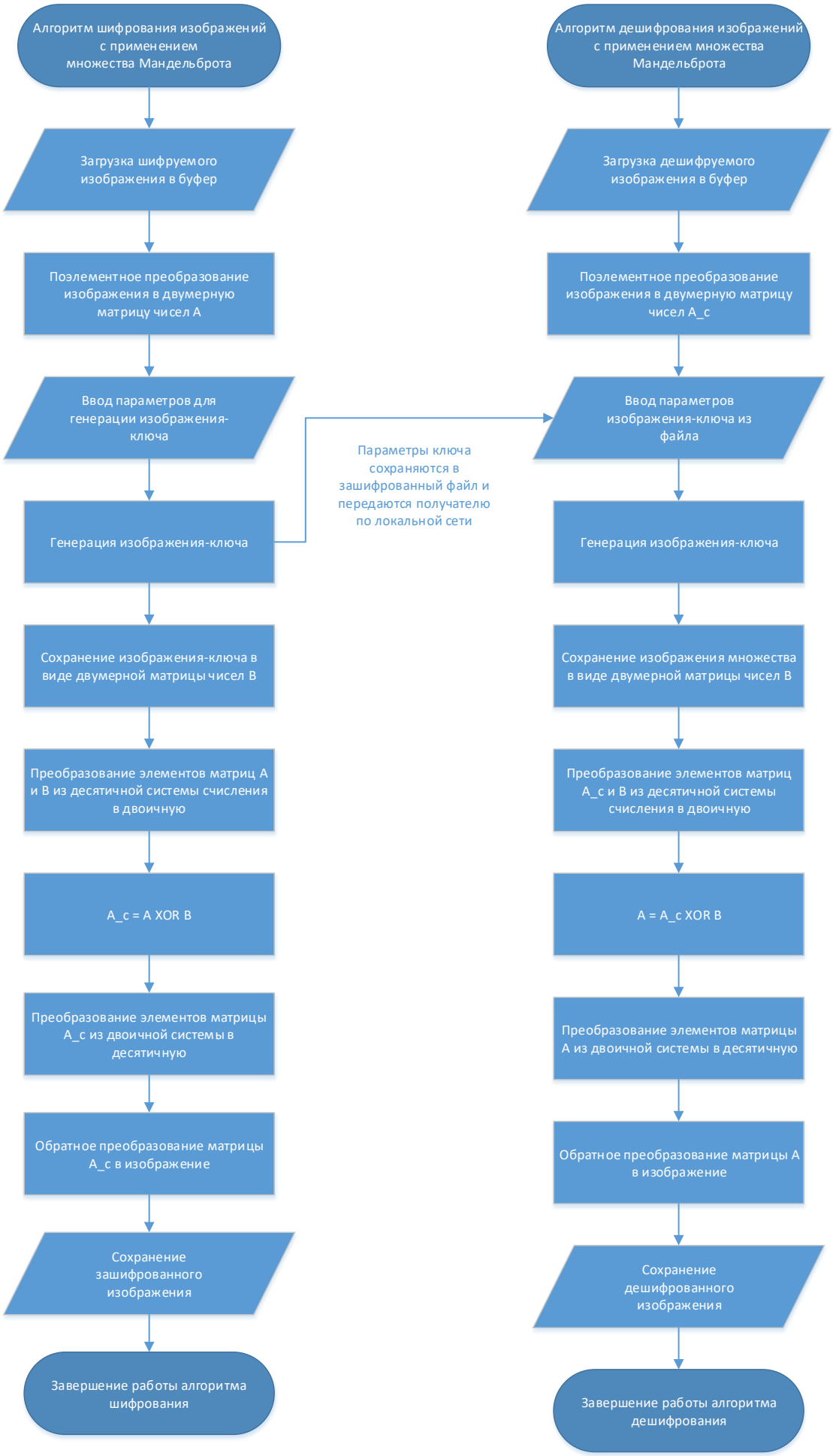
$$(a_{ij} \oplus b_{ij}) \oplus b_{ij} = c_{ij} \oplus b_{ij}$$

Используем свойство ассоциативности и идемпотентности операции XOR:

$$a_{ij} \oplus (b_{ij} \oplus b_{ij}) = c_{ij} \oplus b_{ij} = a_{ij} \oplus 0$$

Используем свойство нулевого элемента**:** $c_{ij} \oplus b_{ij} = a_{ij} \oplus 0 = a_{ij}$

Что и требовалось доказать. Следовательно, изображение множества Мандельброта можно использовать в качестве ключа в следующем симметричном алгоритме шифрования:


Алгоритм шифрования изображений с применением множества Мандельброта
Загрузка шифруемого изображения в буфер
Поэлементное преобразование изображения в двумерную матрицу чисел A
Ввод параметров для генерации изображения-ключа
Генерация изображения-ключа
Сохранение изображения-ключа в виде двумерной матрицы чисел B
Преобразование элементов матриц A и B из десятичной системы счисления в двоичную
A_c = A XOR B
Преобразование элементов матрицы A_c из двоичной системы в десятичную
Обратное преобразование матрицы A_c в изображение
Сохранение зашифрованного изображения
Завершение работы алгоритма шифрования
Параметры ключа сохраняются в зашифрованный файл и передаются получателю по локальной сети
Алгоритм дешифрования изображений с применением множества Мандельброта
Загрузка дешифруемого изображения в буфер
Поэлементное преобразование изображения в двумерную матрицу чисел A_c
Ввод параметров изображения-ключа из файла
Генерация изображения-ключа
Сохранение изображения множества в виде двумерной матрицы чисел B
Преобразование элементов матриц A_c и B из десятичной системы счисления в двоичную
A = A_c XOR B
Преобразование элементов матрицы A из двоичной системы в десятичную
Обратное преобразование матрицы A в изображение
Сохранение дешифрованного изображения
Завершение работы алгоритма дешифрования

Предварительные эксперименты

Проведём сравнение оценок обрабатываемых изображений для различных изображений:

1. Монохромные изображения.
2. Среднецветные изображения (в том числе портреты, пейзажи, интерьеры).
3. Сильноцветные изображения.

Для всех из них получим следующие оценки:

1. Качество дешифрования, определяемое по значению показателя сходства S (SSIM, процент сходства пикселей, коэффициент корреляции) исходного изображения с расшифрованным.
2. Степень различия между исходным и зашифрованным изображением, определяемая по значению показателя сходства S (SSIM, процент сходства пикселей, коэффициенты корреляции общий, в горизонтальном, вертикальном и диагональном направлениях, дисперсия гистограммы распределения яркости на изображении) исходного изображения с зашифрованным.
3. Чувствительность зашифрованного изображения к изменениям ключа, определяемая по количеству пикселей, изменивших значение (NPCR), нормированному среднему изменению интенсивности (UACI) и отношение количества различающихся бит к общему количеству бит изображения (NFC).
4. Среднее время шифрования и дешифрования изображения.

Пусть $D(i,j)$ - результат сравнения пикселей, стоящих на одинаковых позициях, в разных изображениях, n x m – размерность изображения в пикселях, тогда:

$$NPCR = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} D(i,j)}{n \cdot m}$$

$$UACI = \frac{1}{n \cdot m}\sum_{i=1}^{n}\sum_{j=1}^{m}\frac{|a_{1_{ij}} - a_{2_{ij}}|}{255}$$

$$NFC = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} XOR(a_{1_{ij}}, a_{2_{ij}})}{n \cdot m \cdot 8}$$

Рисунок 6. Зашифрованное изображение категории "пейзаж"

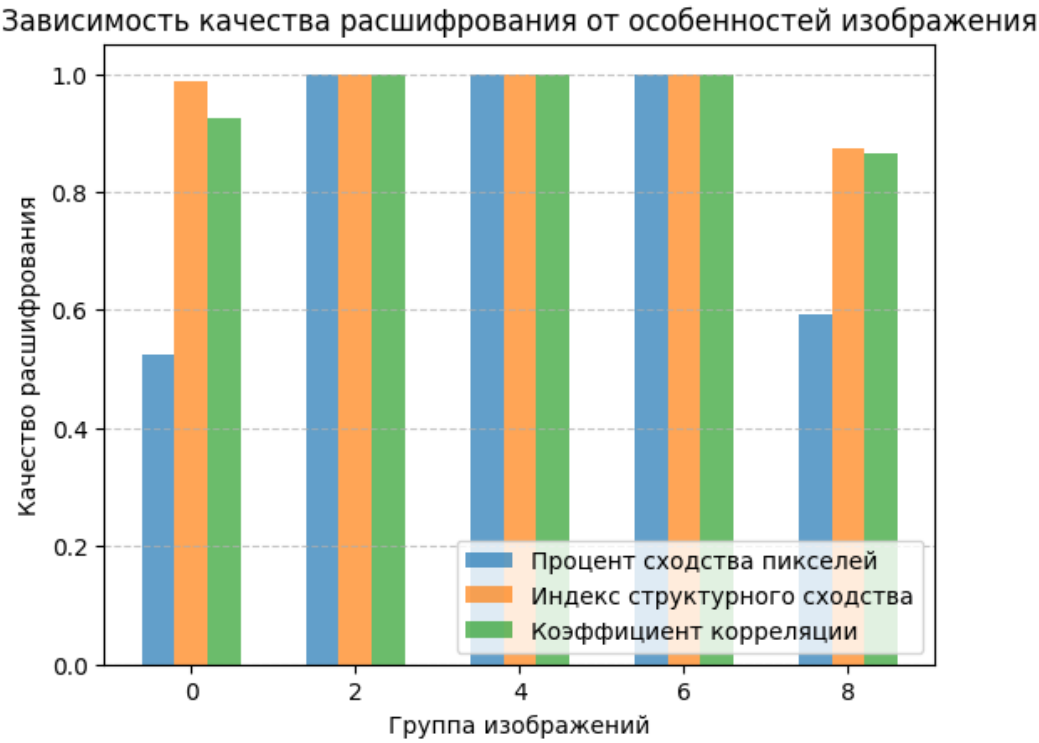


Рисунок 7. Зависимость качества дешифрования от цветности изображения

Следовательно, изображения аномально высокой или, наоборот, низкой цветности могут неправильно расшифровываться, что следует учитывать при выборе изображений для дешифрования.

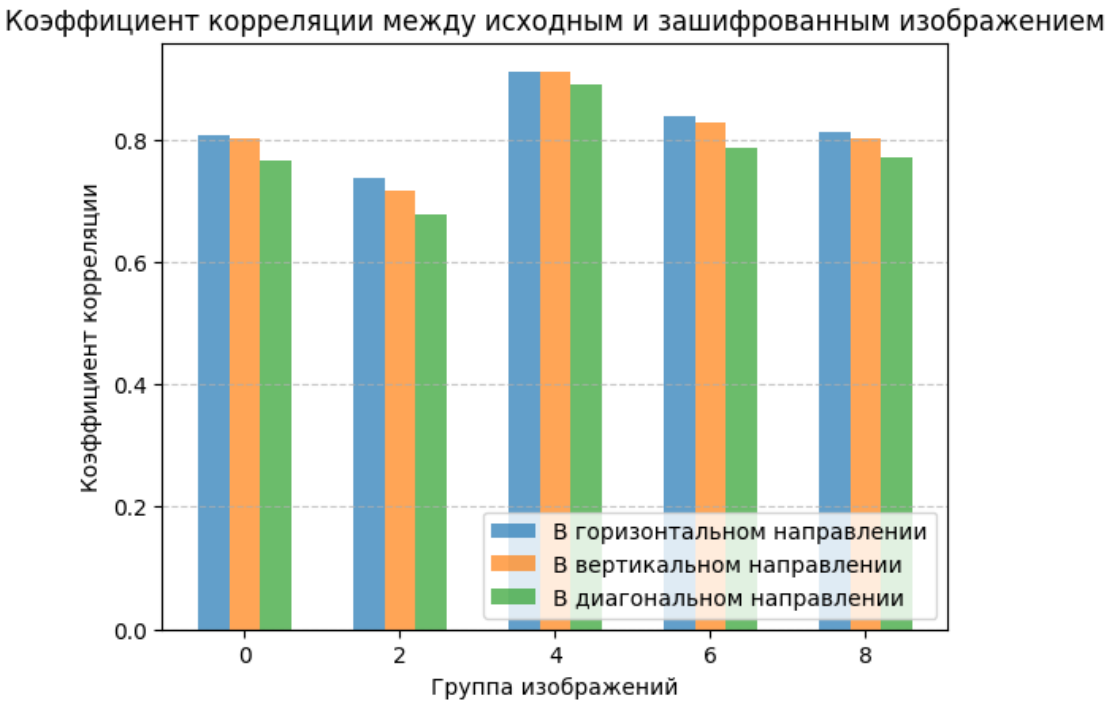


Рисунок 8. Зависимость качества шифрования от особенностей изображения

Результаты этой оценки показывают, что наиболее сильно зашифрованное изображение отличается от исходного в том случае, если мы шифруем портреты:

Рисунок 9. Зашифрованный портрет человека

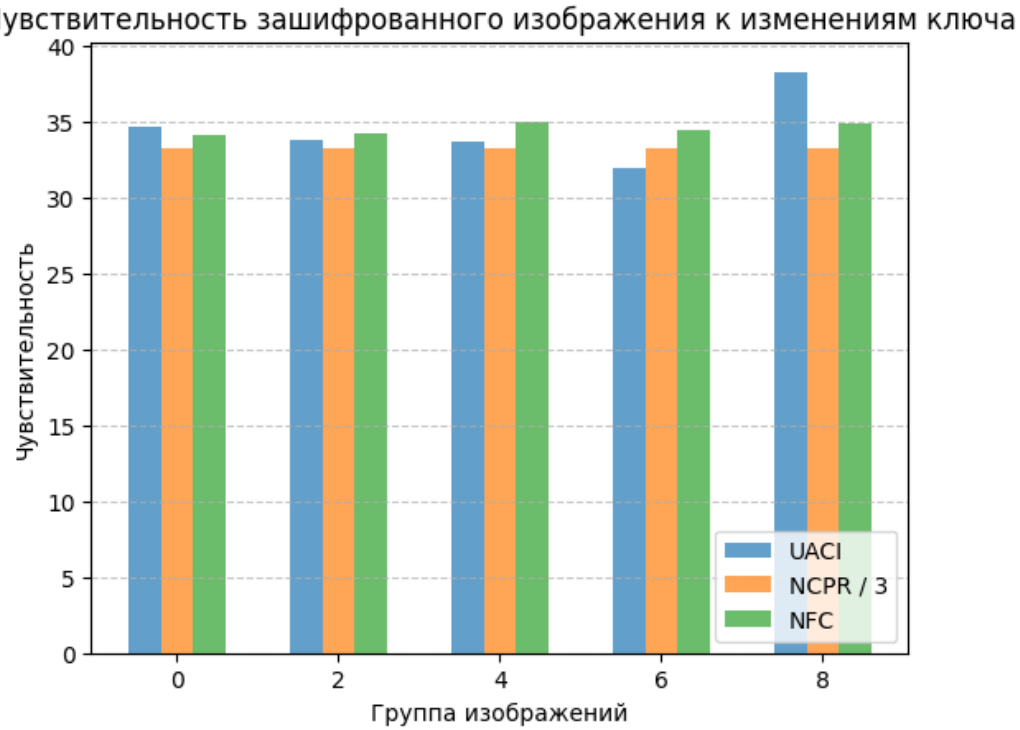


Рисунок 10. Зависимость чувствительности зашифрованного изображения к изменениям ключа от цветности изображения

Здесь различные изображения в целом имеют близкие по значению показатели.

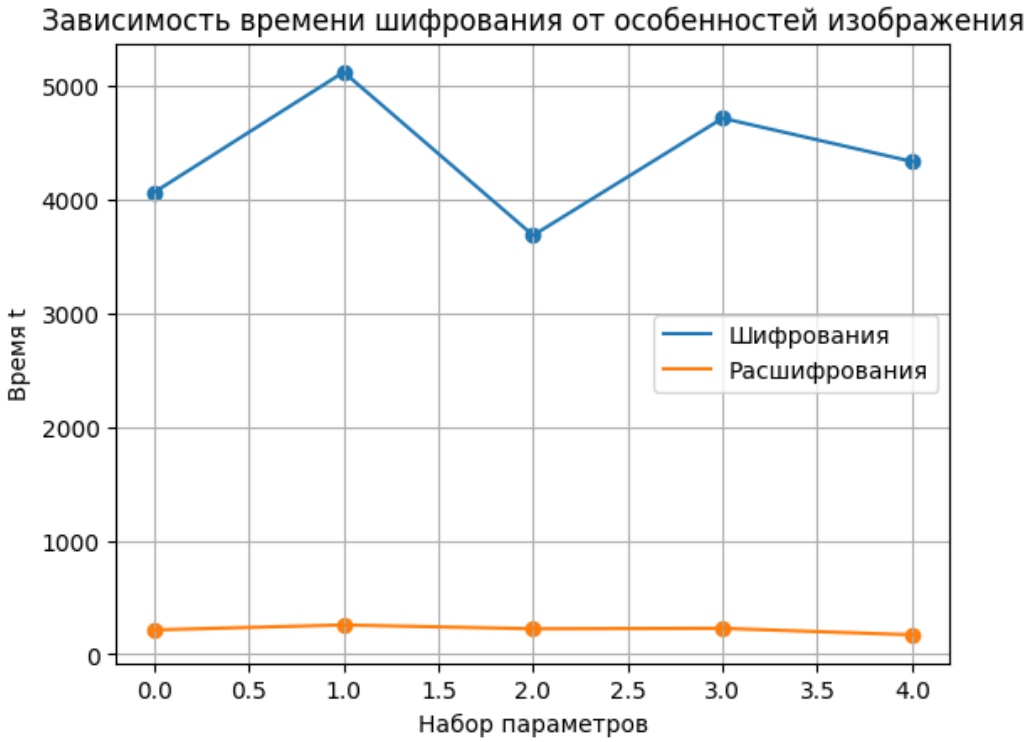


Рисунок 11. Зависимость среднего времени шифрования и дешифрования от особенностей изображения

Исходя из результатов оценки времени шифрования, можно утверждать, что время шифрования не зависит от характера изображения, а является случайной величиной, большая часть которой тратится на генерацию подходящего изображения множества Мандельброта, так как дешифрование происходит во много раз быстрее.

Визуальная оценка полученных зашифрованных изображения показала, что несмотря на значительные изменения структуры изображения, на нём все ещё можно различить отдельные объекты – вплоть до рабочих экранов ПК.

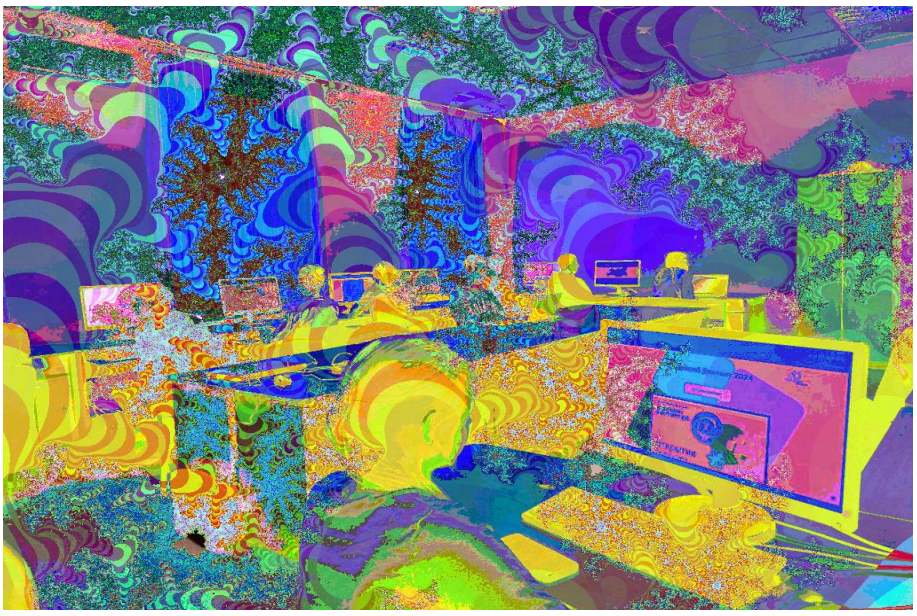


Рисунок 12. Зашифрованное изображение компьютерного класса

Алгоритм сегментации изображения

В качестве способа дополнительного сокрытия объектов на изображении будем разбивать изображение размерности n x m пикселей на $\frac{n}{a} \cdot \frac{m}{b}$ сегментов размерности a x b. Параметры a и b будем подбирать экспериментально таким образом, чтобы сделать объекты на изображении неразличимыми. Случайным образом перемешаем сегменты между собой:

```
Random random = new Random();
for (int i = totalSegments - 1; i > 0; i--) {
    int j = random.nextInt(i + 1);
    int temp = segmentIndices[i];
    segmentIndices[i] = segmentIndices[j];
    segmentIndices[j] = temp;
}
```

Листинг 3. Перемешивание сегментов изображения

Сохраним данные о взаиморасположении сегментов в бинарный файл:

```
Map<Integer, Integer> segmentMapping = new HashMap<>();
for (int i = 0; i < totalSegments; i++) {
    int originalIndex = i; // Исходный индекс сегмента
    int shuffledIndex = segmentIndices[i]; // Перемешанный индекс сегмента
    int segmentX = (shuffledIndex % segmentWidthSize) * segmentWidth;
    int segmentY = (shuffledIndex / segmentWidthSize) * segmentHeight;
    g2d.drawImage(image.getSubimage(segmentX, segmentY, segmentWidth,
                  segmentHeight), (i % segmentWidthSize) * segmentWidth,
                  (i / segmentWidthSize) * segmentHeight, null);
    segmentMapping.put(originalIndex, shuffledIndex);
// Сохраняем исходный индекс как ключ и перемешанный индекс как значение
}
```

Листинг 4. Сохранение разбиения изображения по сегментам

При дешифровании эти данные будут загружены и исходное положение сегментов будет восстановлено. Оценим, насколько хуже стала различима структура зашифрованного изображения:

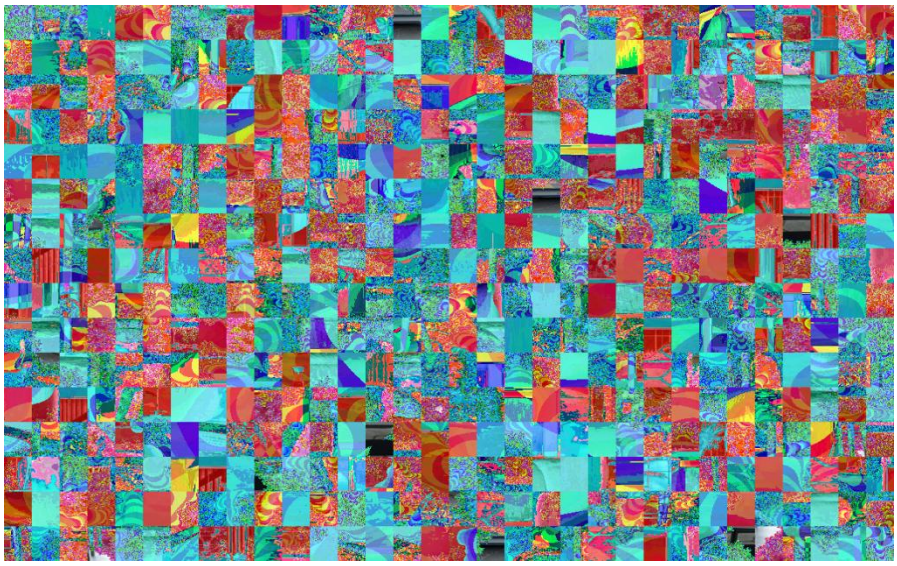


Рисунок 13. Изображение, зашифрованное с использованием дополнительной сегментации

Дополнительные эксперименты

Сравним оценки, полученные в главе «Предварительные эксперименты» без использования сегментации, с аналогичными оценками для расширенного алгоритма шифрования:

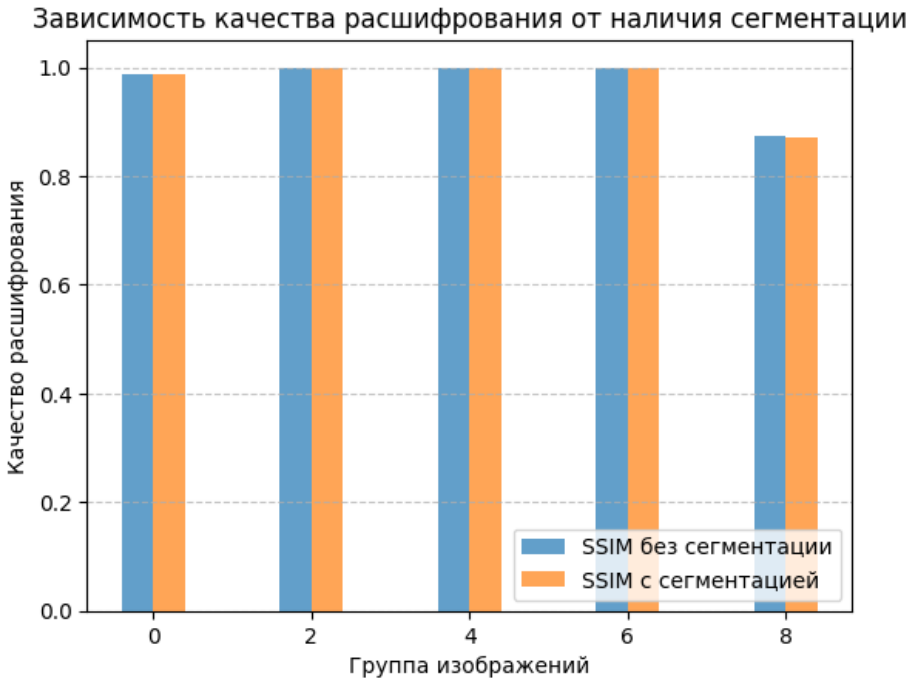


Рисунок 14. Влияние сегментации на качество дешифрования

Как показывают статистические данные, сегментация не оказывает негативного или позитивного влияния на качество дешифрования изображений.

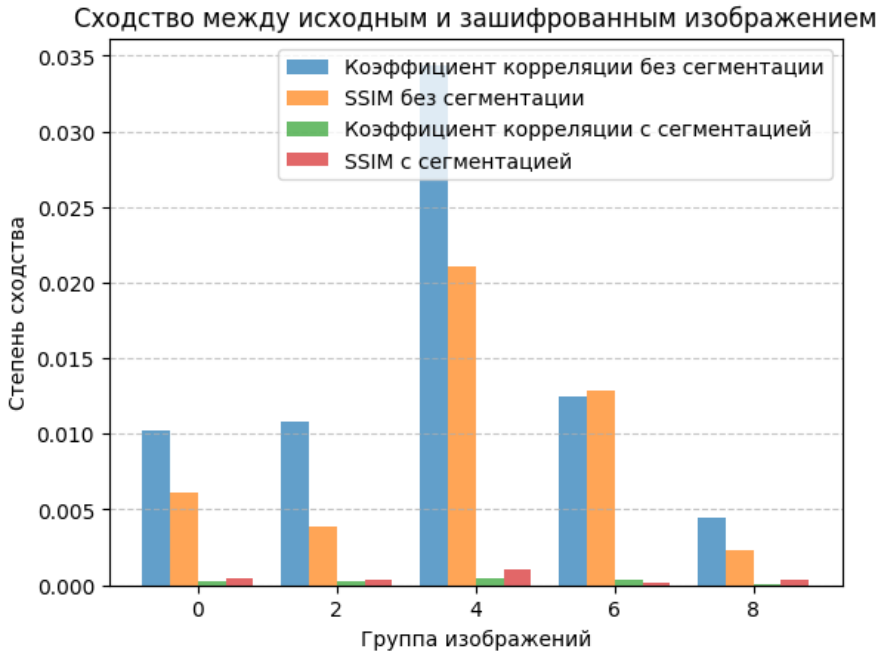


Рисунок 15. Влияние сегментации на показатели качества шифрования

Что и требовалось доказать – анализ показывает, что добавление в алгоритм дополнительной сегментации изображения значительно снижает его сходство с исходным.

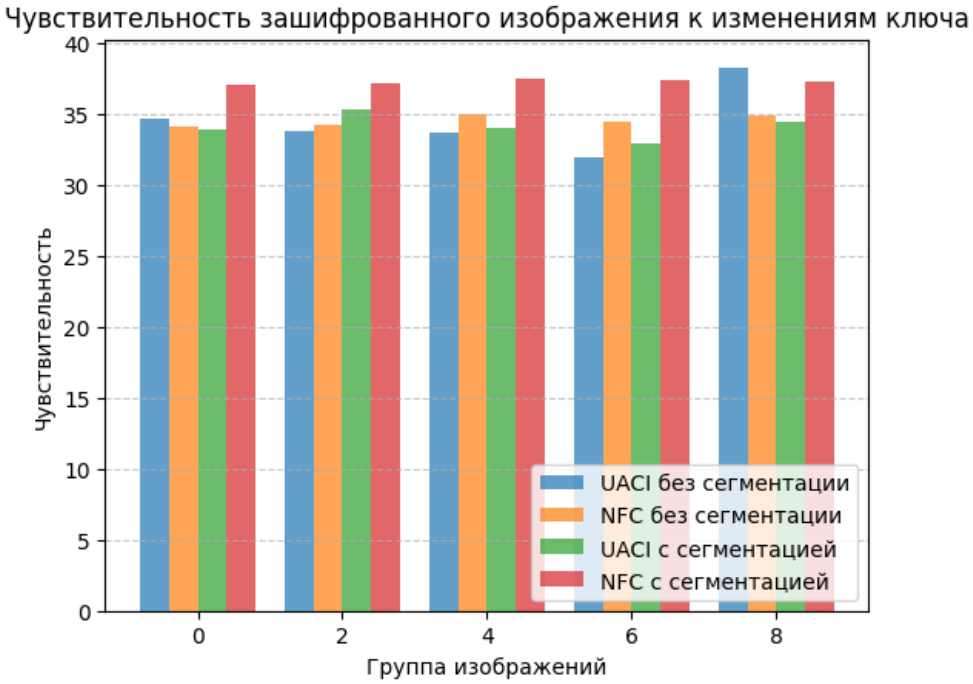


Рисунок 16. Влияние сегментации на чувствительность изображений к ключу

При этом на чувствительность зашифрованного изображения к изменениям ключа сегментация влияет незначительно.

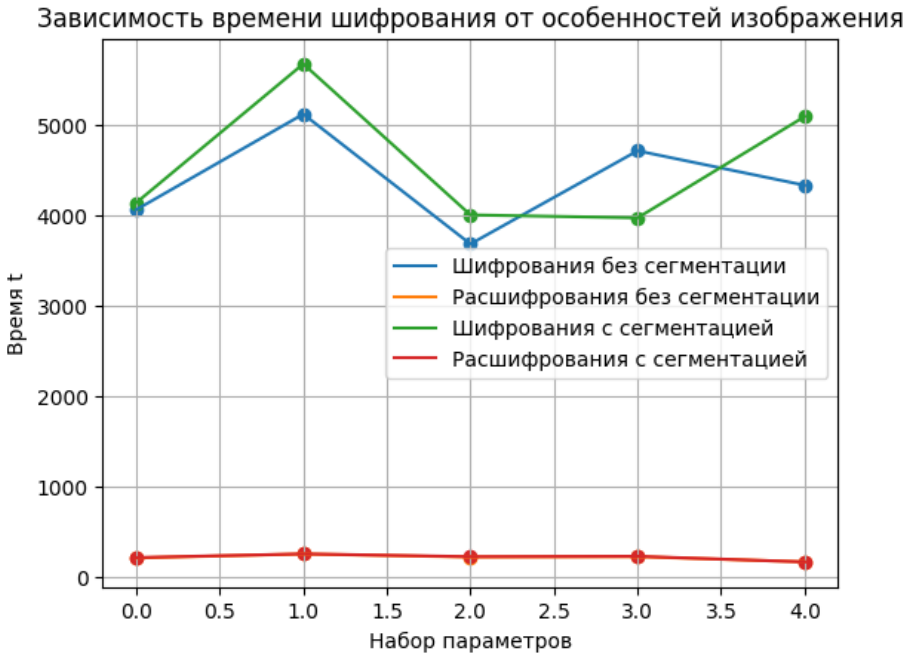


Рисунок 17. Влияние сегментации на время работы алгоритмов

Отметим, что добавление сегментации незначительно изменяет время шифрования изображения чаще в большую сторону, на время дешифрования оно практически не влияет.

Следовательно, в целом добавление сегментации положительно повлияло на показатели предлагаемого алгоритма шифрования и эта функция будет использоваться и далее.

Оценим теперь влияние разрешения изображения на параметры шифрования. Будем обрабатывать следующие разрешения:

1. 640x480;
2. 1024x768;
3. 1366x768 (без дешифрования);
4. 1920x1080 (без дешифрования);

Дешифрование двух последних разрешений не могло быть выполнено из-за фиксированного на данный момент размера сегмента a x b и будет реализовано в дальнейшем. Показатели будем оценивать те же, что и в главе «Предварительные эксперименты».

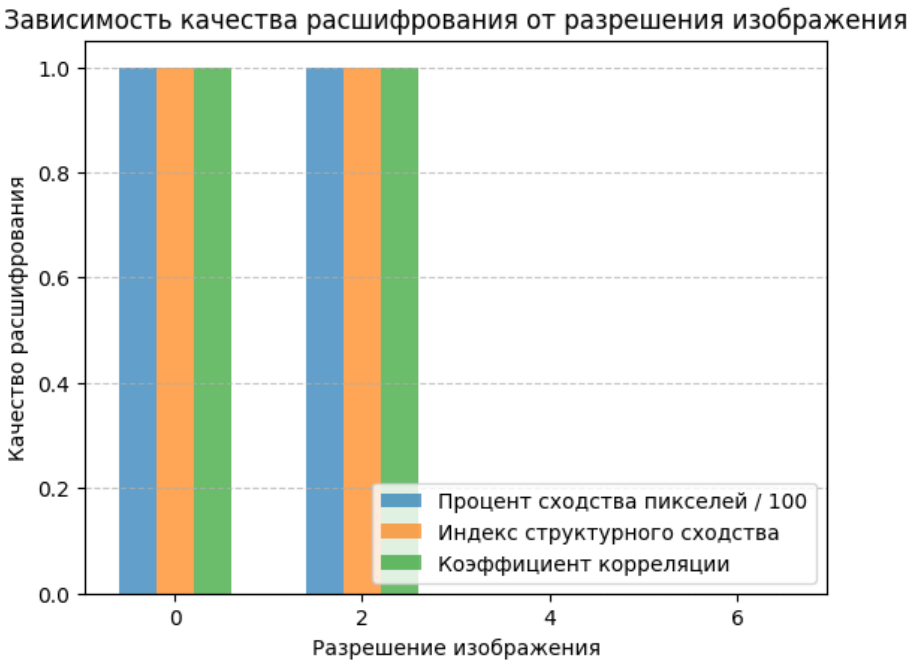


Рисунок 18. Зависимость качества дешифрования от разрешения изображения

Результаты измерений показывают, что качество дешифрования от разрешения не зависит, чего и следовало ожидать.

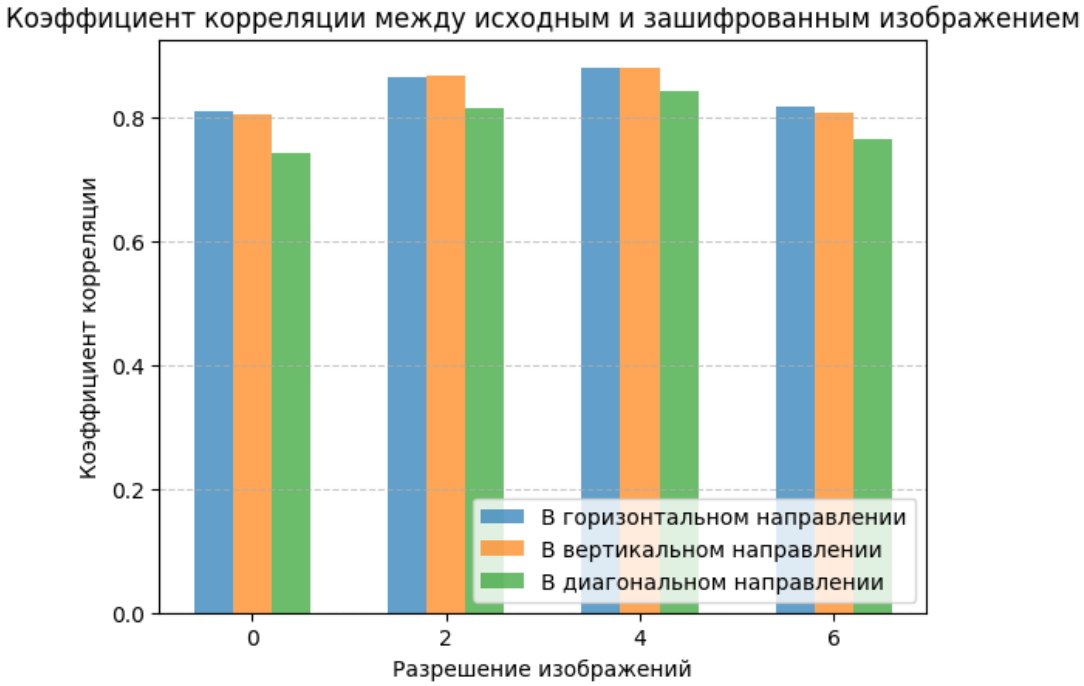


Рисунок 19. Влияние разрешения изображения на качество шифрования

Степень различия между исходным и зашифрованным изображениями, скорее всего, слабо зависит от разрешения.

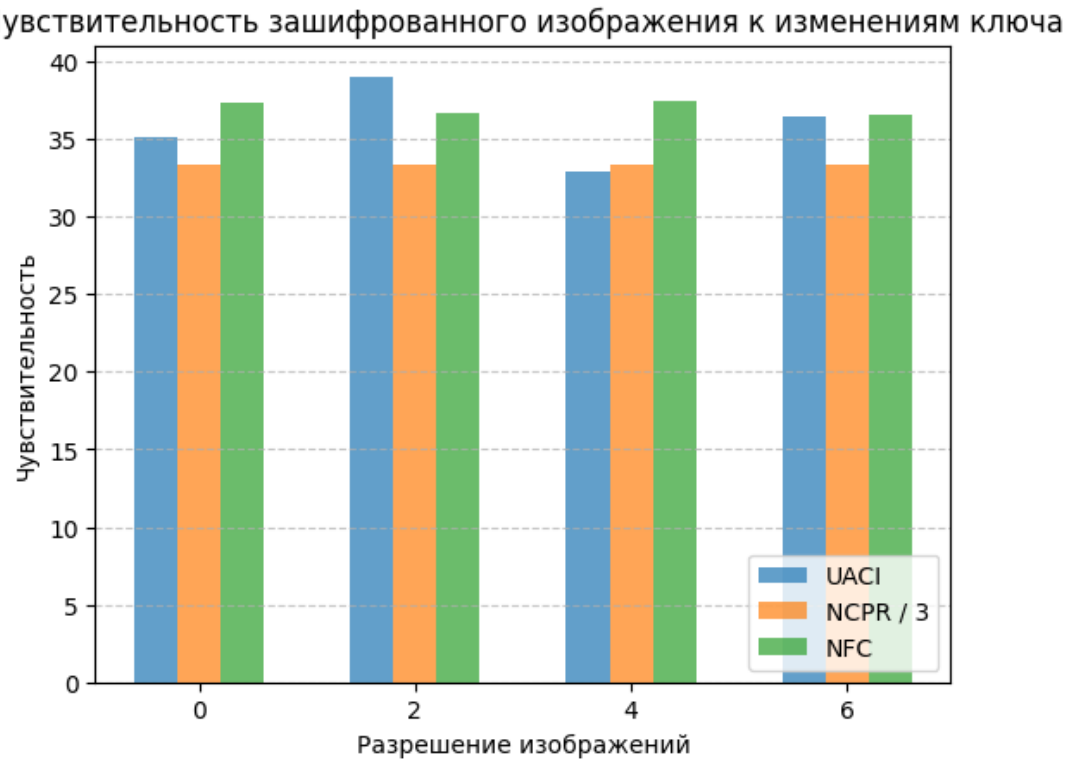


Рисунок 20. Зависимость чувствительности изображения к изменениям ключа от его разрешения

Чувствительность зашифрованного изображения к изменениям ключа при повышении разрешения также сохраняется.

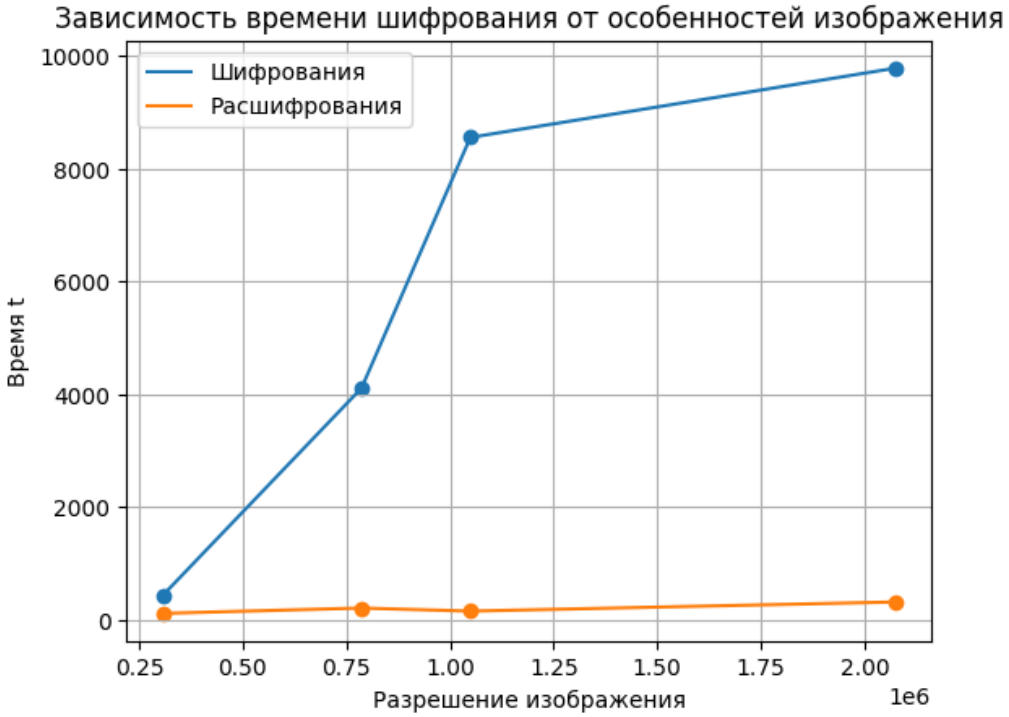


Рисунок 21. Влияние разрешения изображения на время работы алгоритмов

Здесь наибольший интерес представляет нелинейный рост времени шифрования, что, скорее всего, связано с долгой генерацией подходящего изображения множества Мандельброта.

Таким образом, можно утверждать, что ни на что, кроме времени выполнения шифрования, разрешение изображений в значительной степени не влияет.

Сравнение с другими алгоритмами шифрования

В работе [1] был предложен алгоритм шифрования изображений с использованием хаотических отображений, сравнимый с предлагаемым, и представлен ряд показателей, по которым алгоритм сравнивался с AES. Сравним и мы предлагаемый алгоритм с использованием множества Мандельброта с ними, используя показатели различия зашифрованного и исходного изображений, а также показатели чувствительности зашифрованного изображения к изменениям ключа.

Таблица 1.
Значения коэффициентов корреляции между исходным и зашифрованным изображением

Изображение	Алгоритм	$\lvert R_C \cdot 10^3 \rvert$	$\lvert R_h \cdot 10^3 \rvert$	$\lvert R_v \cdot 10^3 \rvert$	$\lvert R_d \cdot 10^3 \rvert$
«Lena.bmp»	предлагаемый	1,26	6,81	7,38	6,36
	хаотический	2,04	1,82	3,97	9,20
	AES	1,91	7,80	9,23	2,34
«Baboon.bmp»	предлагаемый	2,25	5,79	5,42	4,92
	хаотический	1,27	6,74	2,00	2,83
	AES	2,55	7,86	4,81	1,59
«Peppers.bmp»	предлагаемый	3,5	7,39	7,66	7,26
	хаотический	0,88	5,99	2,26	6,67
	AES	3,65	3,97	10,00	1,81

Рассмотрение Таблицы 1 показывает, что предлагаемый алгоритм показывает результаты, сравнимые с аналогичным алгоритмом и стандартным алгоритмом AES, а при некоторых видах изображений – и большие.

Таблица 2.
Показатели различия между исходным и зашифрованным изображением, чувствительности зашифрованных изображений к изменениям ключа

Изображение	Алгоритм	$D \cdot 10^{-3}$	$UACI$	$NPCR \cdot 10^2$
«Lena.bmp»	предлагаемый	0,11	0,336	99,99
	хаотический	1,19	0,335	99,60
	AES	1,10	0,335	99,62
«Baboon.bmp»	предлагаемый	0,08	0,344	99,99
	хаотический	0,29	0,336	99,60
	AES	0,34	0,334	99,57
«Peppers.bmp»	предлагаемый	0,0007	0,336	99,99
	хаотический	0,93	0,335	99,61
	AES	1,16	0,335	99,61

Исходя из данных Таблицы 2 можно утверждать, что по показателям чувствительности предлагаемый алгоритм более эффективен, чем аналогичный и AES.

Таким образом, предлагаемый алгоритм по ряду показателей не уступает аналогичным и может применяться для шифрования изображений, отправляемых по реальным каналам передачи информации.