ГУАП

КАФЕДРА № 51

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

ассистент		А.В. Борисовская
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1
Статистический анализ цифровых изображений

по курсу: МУЛЬТИМЕДИЯ ТЕХНОЛОГИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5912		Б.А. Карханин
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2022

Оглавление

Цель работы:
Изучение способов представления изображений, ознакомление со структурой формата BMP, анализ статистических свойств изображений, а также получение практических навыков обработки изображений.

Описание структуры формата RGB24

1. Заголовок.

2. Данные по пикселям:

Заголовок BITMAPFILEHEADER имеет следующую структуру:

• WORD bfType – описывает тип файла;

• DWORD bfSize – размер файла в байтах;

• WORD bfReserved1 – зарезервировано, должно быть 0;

• WORD bfReserved2 - зарезервировано, должно быть 0;

• DWORD bfOffBits – смещение в байтах от начала заголовка до массива пикселей.

Заголовок BITMAPINFOHEADER имеет следующий вид:

• DWORD biSize – размер структуры в байтах;

• LONG biWidth – ширина изображения в пикселях;

• LONG biHeight – высота изображения в пикселях;

• WORD biPlanes – количество плоскостей изображения;

• WORD biBitCount – количество бит на один пиксель;

• DWORD biCompression – тип сжатия;

• DWORD biSizeImage – размер изображения в байтах;

• LONG biXPelsPerMeter – горизонтальное разрешение в пикселях на метр;

• LONG biYPelsPerMeter – вертикальное разрешение в пикселях на метр;

• DWORD biClrUsed – размер палитры;

• DWORD biClrImportant – число значимых элементов палитры.

После заголовка следует информация о пикселях. Данная информация представлена в виде одномерного массива, в котором значения, относящиеся к отдельным пикселям, записаны строка за строкой. Строки могут идти как

снизу-вверх, так и сверху-вниз. Ширина строки в байтах должна быть выравнена по границе двойного слова (32 бита), т.е. должна быть кратна числу 4. При необходимости для выполнения этого условия в конце каждой строки добавляются дополнительные байты (от 1 до 3-х), значения которых несущественны. Каждый пиксель в формате RGB24 представляется тремя байтами. Первый содержит значение компоненты B, второй — G, третий — R. Структура имеет название RGB.

При исследовании использовались поля:

• biWidth, biHeight – для определения количества пикселей в картинке и последующего их использования;

Выполнение работы:

1. Согласование BMP-файла с преподавателем

Рис. 1 Исходное изображение

2. Загрузка исходного изображения в память указанного BMP-файла.

Чтение BMP файла в память выполняется с помощью функции loading_bmp. Функция получает файл изображения, в процессе своей работы создает две структуры заголовка и массив пикселей.

3. Выделение компонент R, G и B.

Создается три файла (Red. bmp, Green. bmp, Blue. bmp) по следующему правилу: данные, не относящиеся к соответствующей компоненте,

обнуляются. Таким образом, получаем 3 файла с соответствующими цветами.

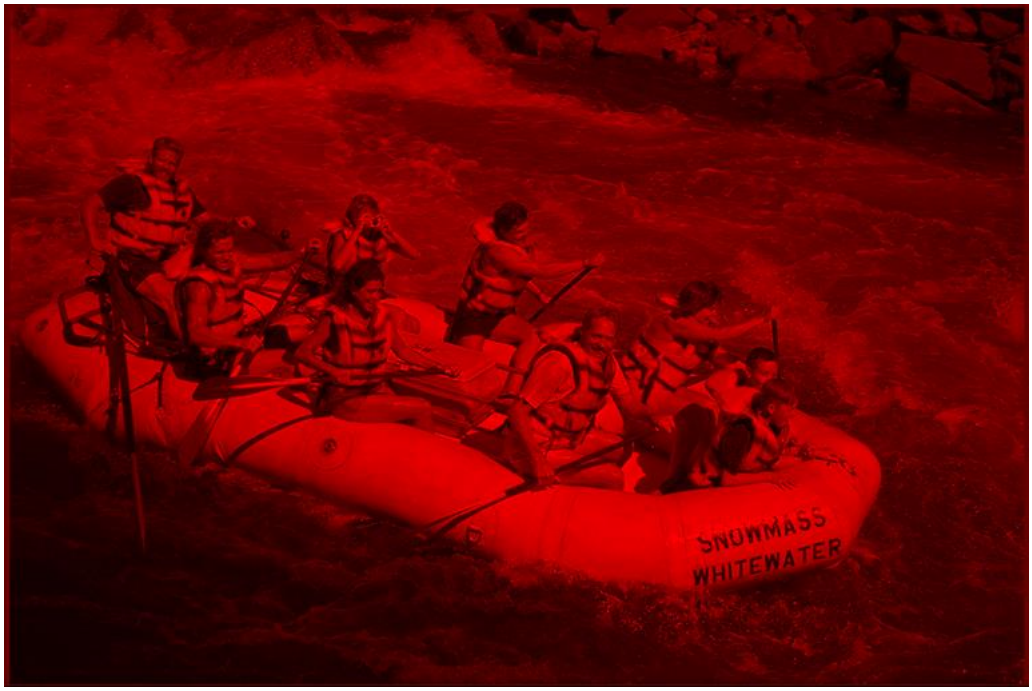

Рис. 2 Выделение компоненты R

Рис. 3 Выделение компоненты G

Рис. 4 Выделение компоненты B

4. Анализ корреляционных свойств компонент R, G, B

а)

Коэффициент корреляции указывает на наличие связи и принимает значения от –1 до +1. Например, при анализе связи между двумя переменными получен коэффициент парной корреляции, равный *–1*. Это означает, что между переменными существует точная обратная линейная зависимость, а если она близка к нулю, то они менее зависимы.

```
correlation R-G = 0.861316
correlation R-B = 0.456784
correlation B-G = 0.710742
```

Рис. 5 полученные коэффициенты корреляции

Как видно по полученным данным, компоненты R и G более зависимы друг с другом, чем другие пары компонент.

б)

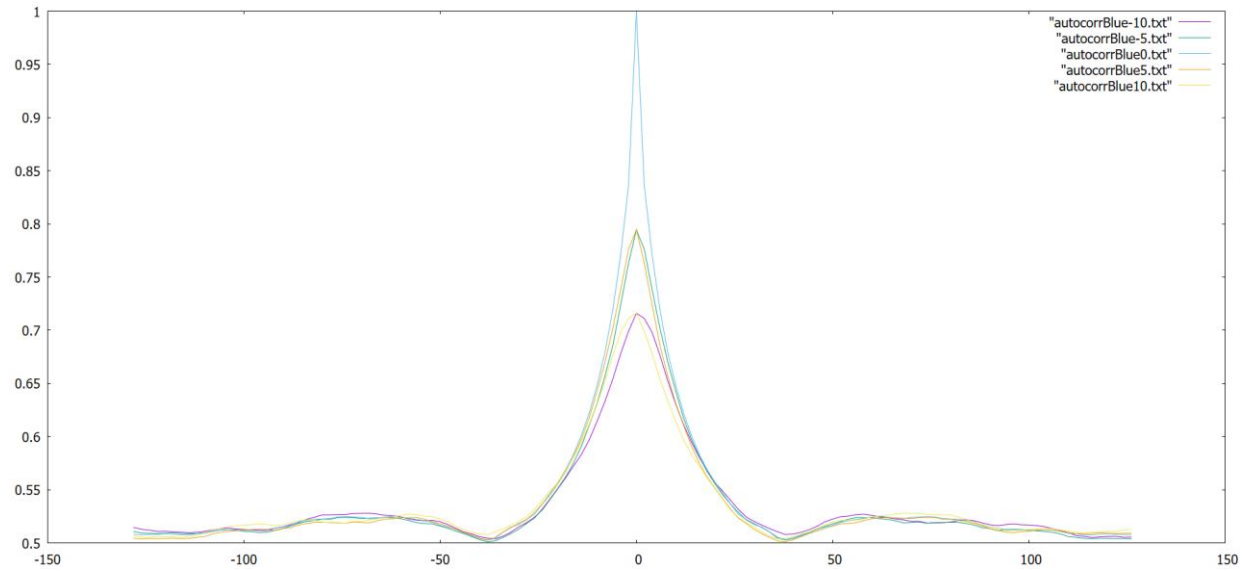

Рисунок 6. Графики автокорреляционной функции компоненты Blue

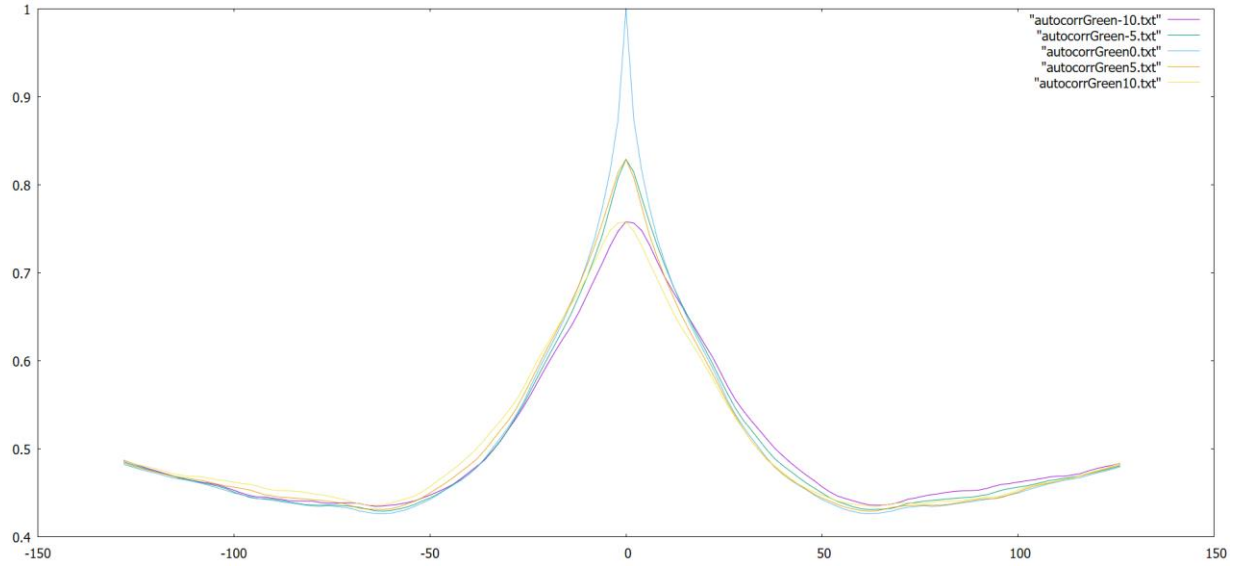

Рисунок 7. Графики автокорреляционной функции компоненты Green

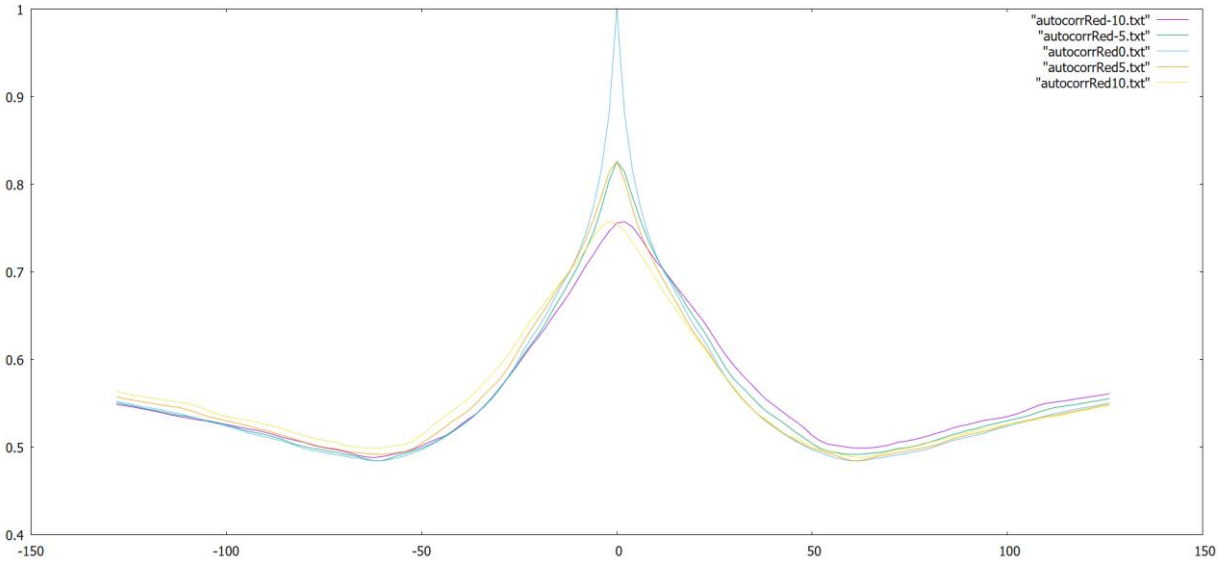

Рисунок 8. Графики автокорреляционной функции компоненты Red

По этим графикам видно, что при $y = 0$ и $x = 0$, значения автокорреляционной функции равно 1 для всех компонент, так как исходная картинка не сдвигается, и находится зависимость исходной картинки с ее точной копией. Также, при изменении y на одинаковые значения в отрицательную и положительную сторону, максимальное значение корреляции будет меньше 1 и одинакова для положительного и отрицательного y, так как картинка смещается на одно и то же количество пикселей. При удалении от $x = 0$ значения коэффициента уменьшается, так как картинка будет сдвигаться на большее количество пикселей.

5. Преобразование данных из RGB системы в YCbCr.

Преобразование данных из RGB в YCbCr производится по формуле:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B; \\ C_b &= 0.5643(B - Y) + 128; \\ C_r &= 0.7132(R - Y) + 128. \end{aligned}$$

Формат YCbCr позволяет получить меньшую зависимость компонент (меньшую корреляцию).

Результат вычисления программы корреляции между компонентами формата YCbCr:

```
correlation Y-Cb = -0.545834
correlation Y-Cr = 0.113824
correlation Cb-Cr = -0.629451
```

Рисунок 9. Значения коэффициентов корреляции для YCbCr

Как можно видеть, зависимость между компонентами формата YCbCr меньше, нежели между компонентами формата BMP. Это позволяет избежать избыточности информации при сжатии, а также снизить риск повреждения всей картинки при повреждении/изменении одной компоненты.

6. Выделение содержимого компонент Y, Cb, Cr.

Создается три файла (contentY.bmp, contentCb.bmp, contentCr.bmp) по следующему правилу: значение компоненты для каждого пикселя сохраняется во всех трех байтах, соответствующих позициям R, G и B этого пикселя.

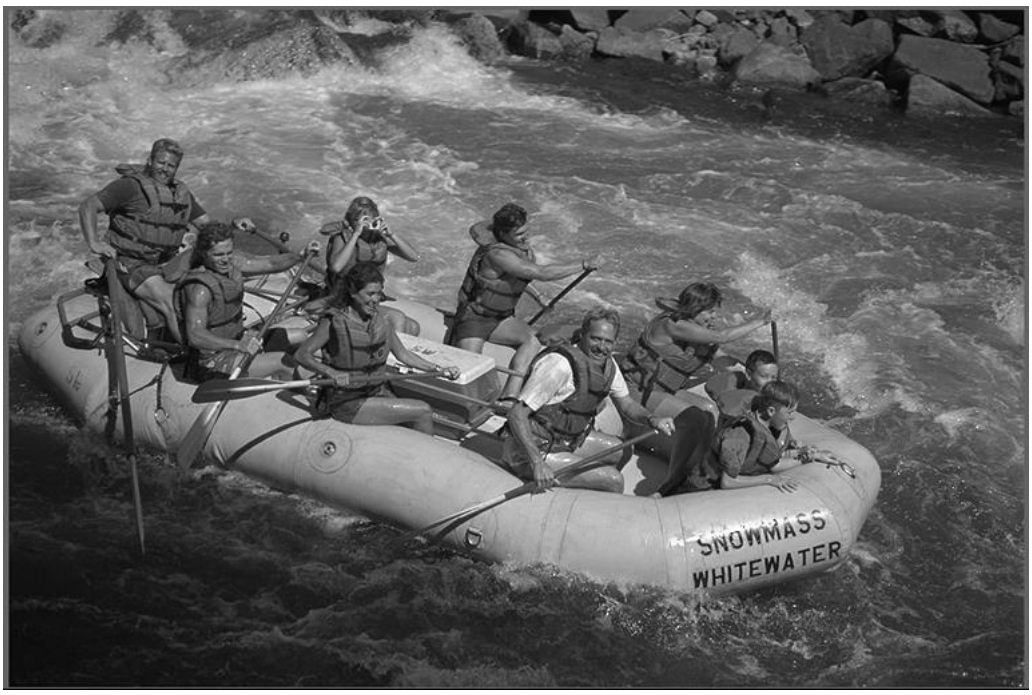

Рисунок 10. Выделение компоненты Y

Рисунок 11. Выделение компоненты Cr

Рисунок 12. Выделение компоненты Cb

7. Обратное преобразование данных из формата YCbCr в RGB

Обратное преобразование выполнятся по формулам:

$$\begin{aligned} G &= Y - 0.714(C_r - 128) - 0.334(C_b - 128); \\ R &= Y + 1.402(C_r - 128); \\ B &= Y + 1.772(C_b - 128). \end{aligned}$$

Рисунок 12. Исходное изображение

Рисунок 13. Восстановленное изображение

Преобразование форматов является необратимым, так как выполняется с помощью округления. Из-за этого возможен выход восстановленных значений за границы исходного диапазона, поэтому необходимо выполнить операцию клиппирования, по формулам ниже:

$$\mathrm{Sat}(x, x_{min}, x_{max}) = \begin{cases} x_{min} & \text{, если } x < x_{min}; \\ x_{max} & \text{, если } x > x_{max}; \\ x & \text{, во всех иных случаях.} \end{cases}$$

Значение peak signal-to-noise ratio (PSNR, сигнал/шум) вычисляется по формуле:

$$PSNR = 10 \lg \frac{WH(2^L - 1)^2}{\sum_{i=1}^{H} \sum_{j=1}^{W} \left(I_{i,j}^{(A)} - \hat{I}_{i,j}^{(A)}\right)^2}.$$

Полученные программно значения PSNR:

```
#7
PSNR R = 49.4788
PSNR G = 48.3461
PSNR B = 63.4701
```

Рисунок 14. PSNR for RGB

По этим значениям видно, что при восстановлении интенсивности компонент изменились, так как при прямом и обратном преобразовании были использованы дробные числа, которые отсекались при их восстановлении в RGB. Однако видимых различий между восстановленным и исходным изображении не обнаружено. Это связано с тем, что значение PSNR выше минимального значения, при котором изображение восстанавливается без потерь примерно равное 35.

8-10. Децимация в 2 раза

Децимация производится только на отдельно записанных компонентах Cb и Cr, так как в изображениях формата YCbCr они не несут основную информационную нагрузку (за это отвечает в основном яркостная компонента Y), можно в несколько раз уменьшить информацию, передаваемую данными компонентами без серьёзных потерь в качестве изображения. Для этого существует понятие "децимация". В работе тестируется децимация двух видов:

а) Исключение строк и столбцов с нечётными номерами.

Значения PSNR в данном случае, при децимации в 2 раза, следующие:

```
Decimation = 2 (a)
YCbCr
PSNR Cb = 39.4418
PSNR Cr = 39.3089
RGB
PSNR R = 36.3659
PSNR G = 42.694
PSNR B = 35.4989
```

Рисунок 15. PSNR при децимации в 2 раза, путем исключения четных строк и столбцов

Глядя на данные, видно, что PSNR снизился, что свидетельствует о частичной потере данных, но не критичной.

Рисунок 16. Сравнение исходного (слева) изображения и восстановленного (справа) при децимации в 2 раза путем исключения четных строк и столбцов.

Как можно заметить, явных искажений не наблюдается даже при наблюдении в масштабе 395%.

b) Среднее арифметическое смежных элементов (соседи по диагонали не рассматриваются, только верхние, нижние, левые и правые).

Значения PSNR в данном случае, при децимации в 2 раза, следующие:

```
Decimation = 2 (b)
YCbCr
PSNR Cb = 42.3221
PSNR Cr = 42.2883
RGB
PSNR R = 39.195
PSNR G = 45.2891
PSNR B = 37.9662
```

Рисунок 17. PSNR при децимации в 2 раза, путем нахождения среднего арифметического смежных элементов.

Значения PSNR при данном типе децимации больше, чем в пункте а), так происходит из-за того, что учитываются смежные пиксели.

Рисунок 18. . Сравнение исходного (слева) изображения и восстановленного (справа) при децимации в 2 раза путем нахождения среднего арифметического соседних элементов.

Опять же, видимых изменений не наблюдается.

11. Децимация в 4 раза

а) Исключение строк и столбцов с нечётными номерами.

```
Decimation = 4 (a)
YCbCr
PSNR Cb = 33.0663
PSNR Cr = 32.6475
RGB
PSNR R = 29.7642
PSNR G = 37.2281
PSNR B = 29.3537
```

Рисунок 19. PSNR при децимации в 4 раза, путем исключения четных строк и столбцов

По полученным показателем PSNR видно, что потери данных значительно выше, по сравнению с децимацией в 2 раза.

Рисунок 20. Сравнение исходного (слева) изображения и восстановленного (справа) при децимации в 4 раза путем исключения четных строк и столбцов.

Картинка подтверждает значительную потерю данных: видно, что красная компонента сильно преобладает на восстановленной картинке, что является некорректным.

b) Среднее арифметическое смежных элементов (соседи по диагонали не рассматриваются, только верхние, нижние, левые и правые).

```
Decimation = 4 (b)
YCbCr
PSNR Cb = 36.9641
PSNR Cr = 36.7306
RGB
PSNR R = 33.786
PSNR G = 40.9825
PSNR B = 32.8765
```

Рисунок 21. PSNR при децимации в 4 раза, путем нахождения среднего арифметического смежных элементов

Значения PSNR при данном типе децимации больше, чем в пункте а), но меньше, чем при децимации в 2 раза.

Рисунок 22. Сравнение исходного (слева) изображения и восстановленного (справа) при децимации в 4 раза путем нахождения среднего арифметического соседних элементов.

Картинка повреждена, но красная компонента в этом случае не столь ярко выражена, как в пункте а)

12. Построение гистограмм частот компонент R, B, G, Y, Cb и Cr.

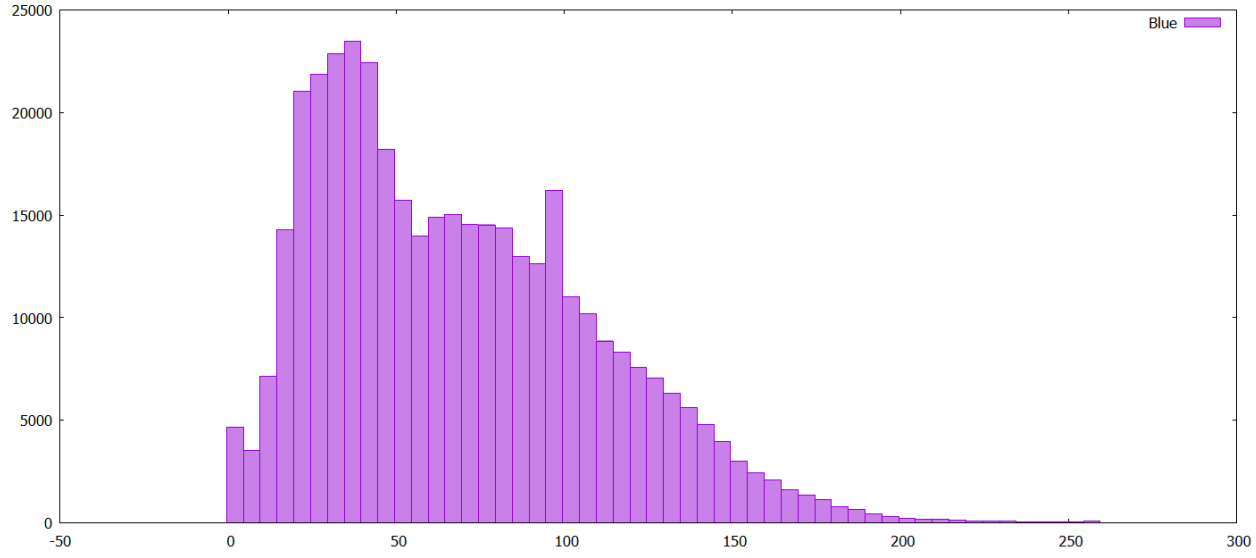

Рисунок 23. Гистограмма компоненты B

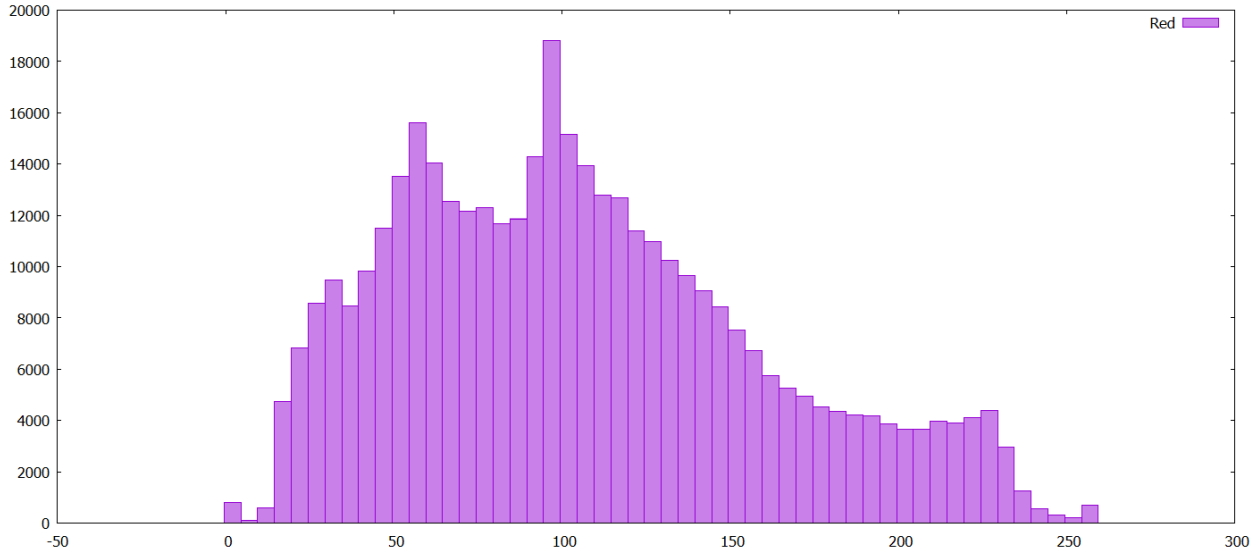

Рисунок 24. Гистограмма компоненты R

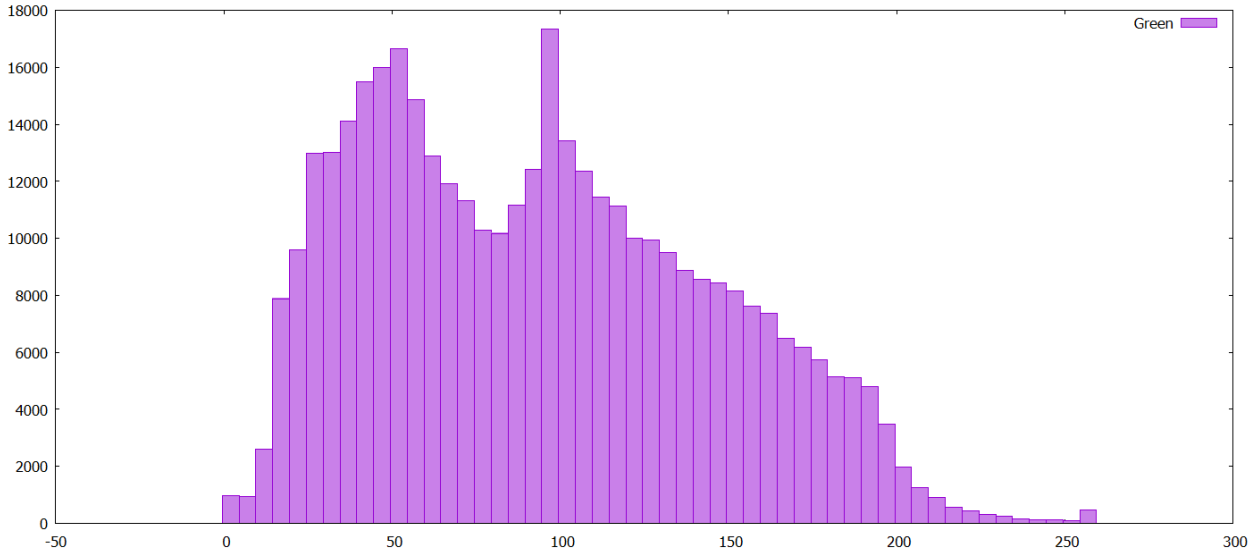

Рисунок 25. Гистограмма компоненты G

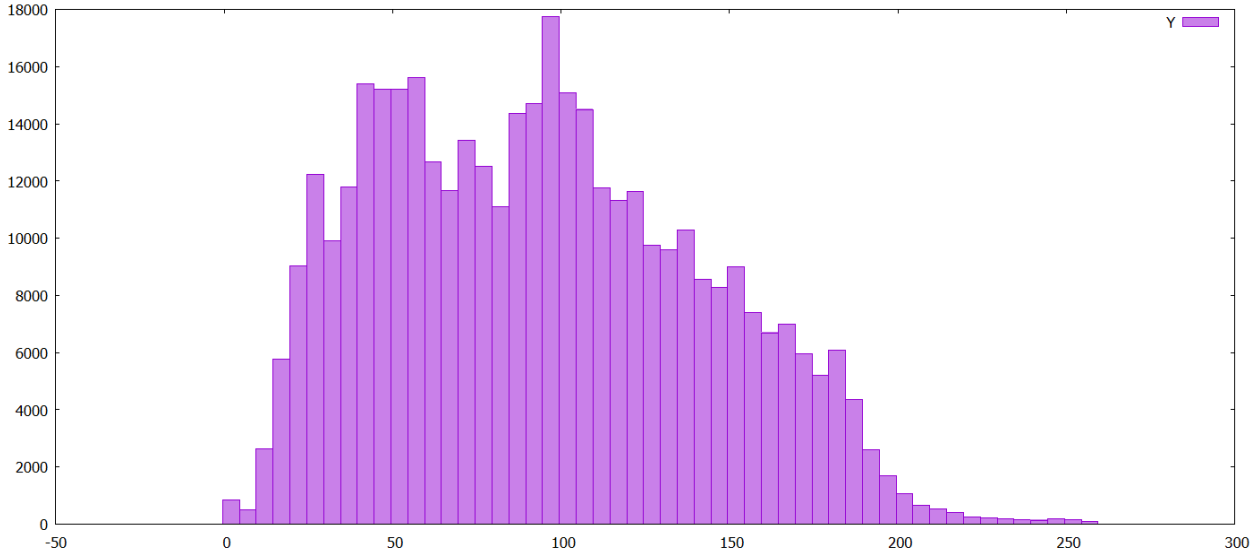

Рисунок 26. Гистограмма компоненты Y

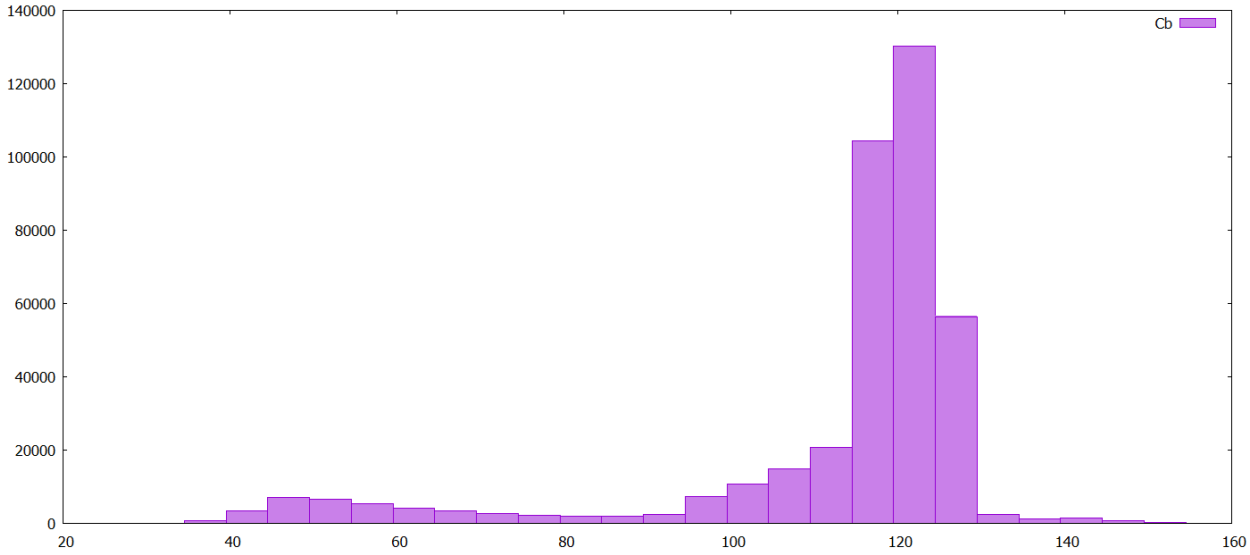

Рисунок 27. Гистограмма компоненты Cb

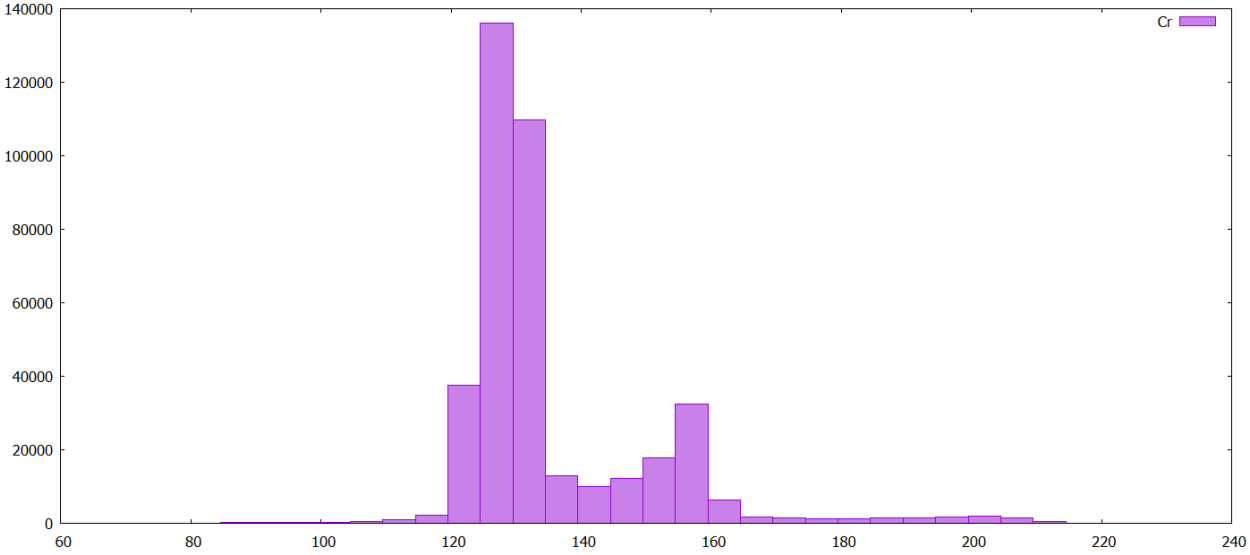

Рисунок 28. Гистограмма компоненты Cr

По данным гистограмм видно, что компоненты R, G, B, Y распределены более равномерно по области значений, что свидетельствует о том, что эти компоненты будут сжиматься хуже, нежели компоненты Cb и Cr, которые более сосредоточены в одной области.

13. Оценка энтропии

Энтропия для каждой компоненты вычисляется по следующей формуле:

$$\hat{H}(X) = -\sum \hat{p}(x) \cdot \log_2 \hat{p}(x)$$

Рассчитанные программно значения:

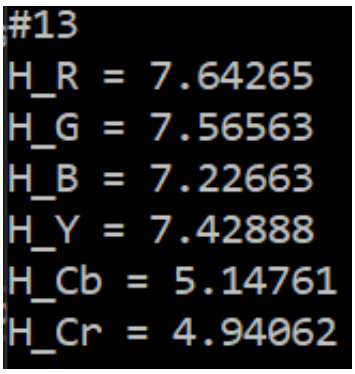

Рисунок 29. Оценка энтропии при поэлементном независимом сжатии компонент R, G, B, Y , Cb и Cr

Энтропия меньше всего у компонент Cb и Cr, как и было видно по гистограммам.

14-15. Количественный анализ эффективности разностного кодирования, которого используется в алгоритмах класса DPCM.

Массивы DPCM формируются по следующему правилу:

$$d_A^{(r)}(i,j) = a(i,j) - f^{(r)}(i,j)$$

где a(i, j) - значение пикселя в компоненте A на позиции (i, j), а $f^{(r)}$(i, j) - правило с номером r, по которому вычисляется предсказание пикселя на позиции (i, j). В исследовании используются следующие правила предсказания:

1 - сосед слева (на позиции (i, j-1));

2 - сосед сверху (на позиции (i-1, j));

3 - сосед сверху слева (на позиции (i-1, j-1));

4 - среднее арифметическое трех соседей - сверху, слева и сверху слева.

1) Массив DPCM, построенный по правилу 1

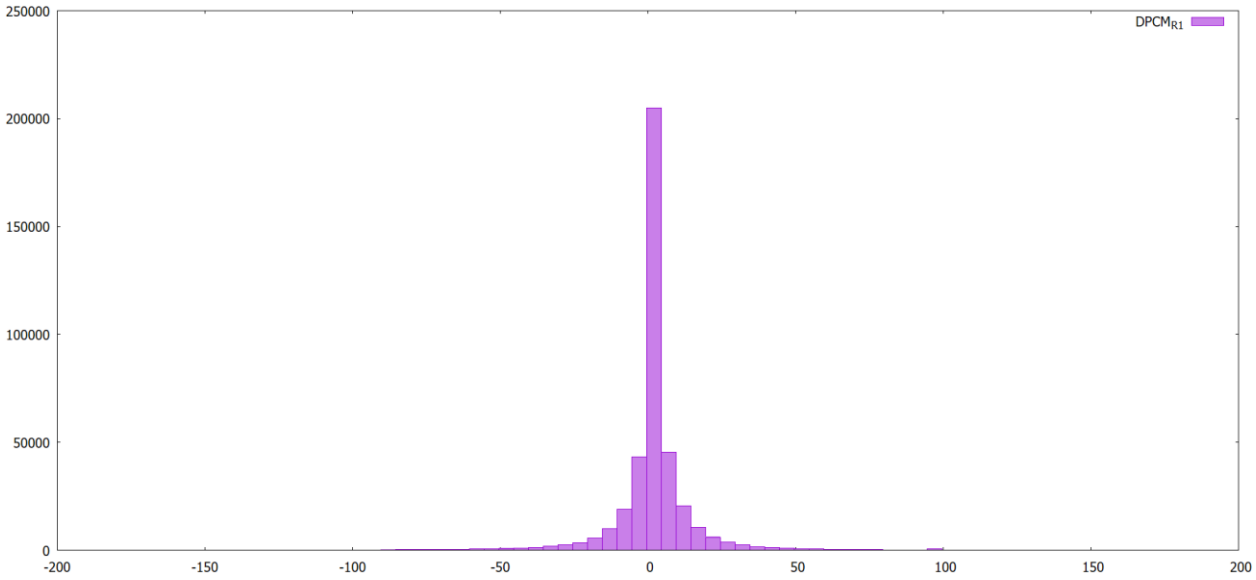

Рисунок 30. Гистограмма компоненты R, DPCM по 1 правилу

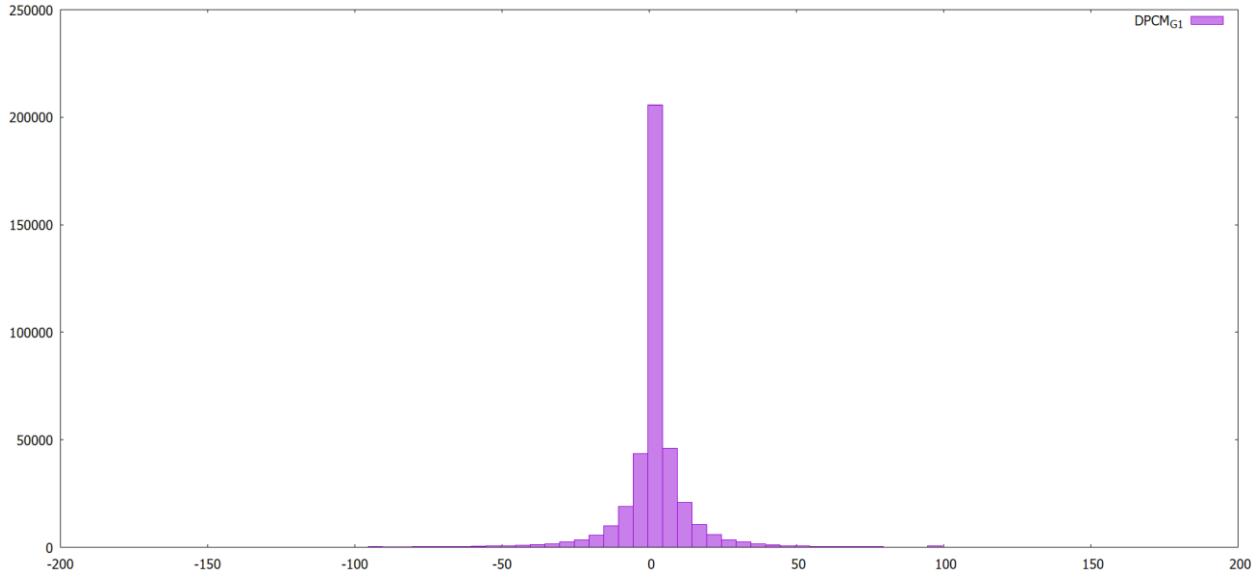

Рисунок 31. Гистограмма компоненты G, DPCM по 1 правилу

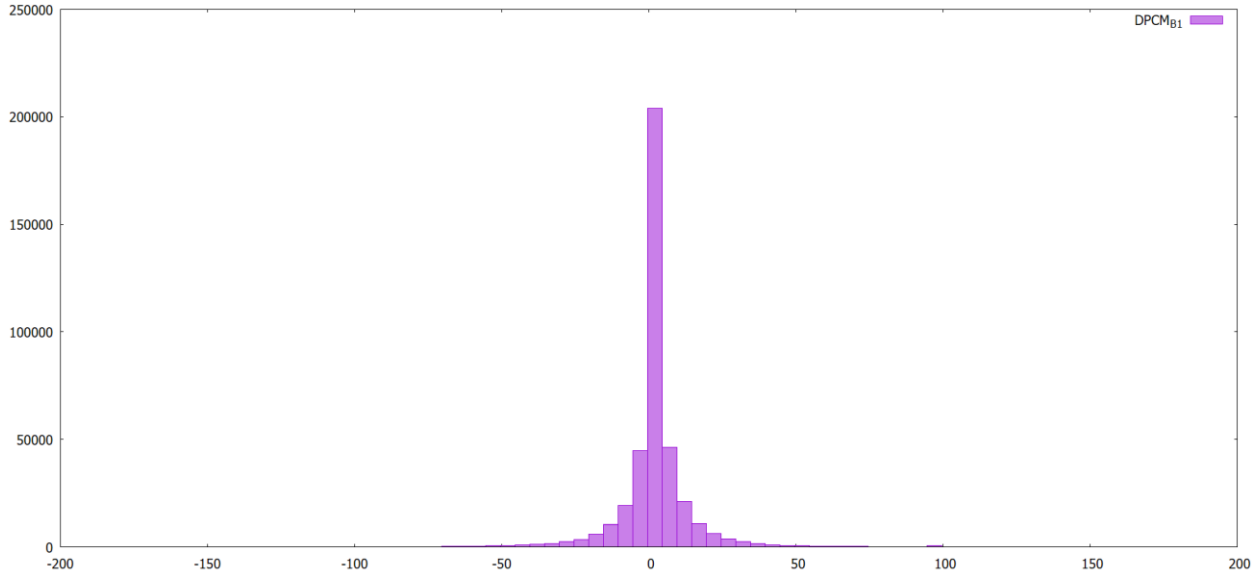

Рисунок 32. Гистограмма компоненты B, DPCM по 1 правилу

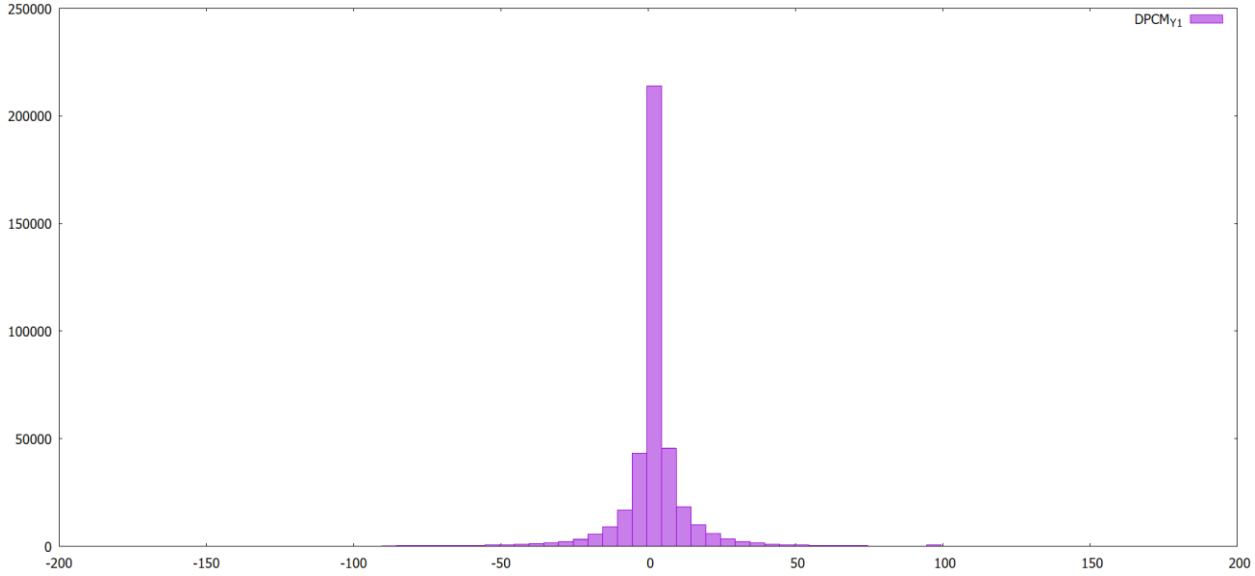

Рисунок 33. Гистограмма компоненты Y, DPCM по 1 правилу

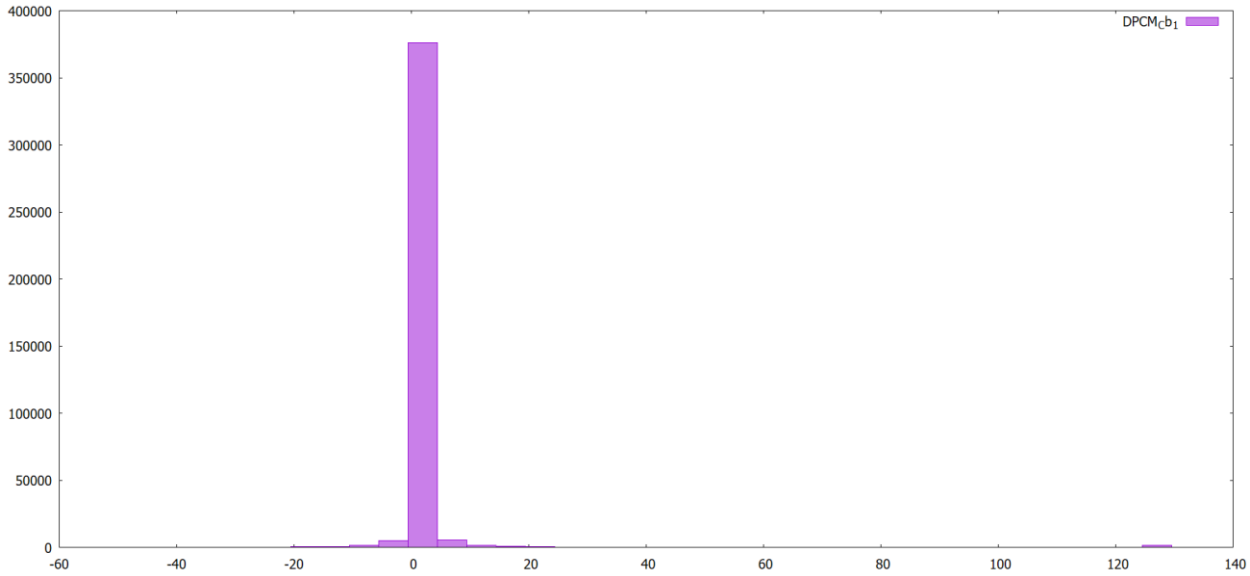

Рисунок 34. Гистограмма компоненты Cb, DPCM по 1 правилу

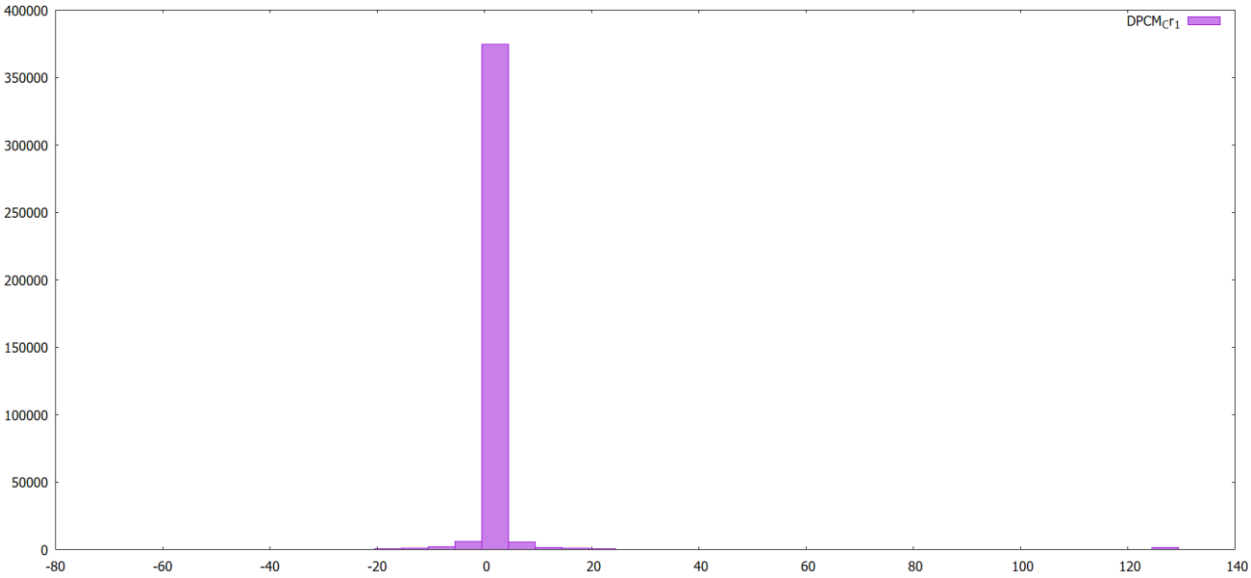

Рисунок 35. Гистограмма компоненты Cr, DPCM по 1 правилу

2) Массив DPCM, построенный по правилу 2

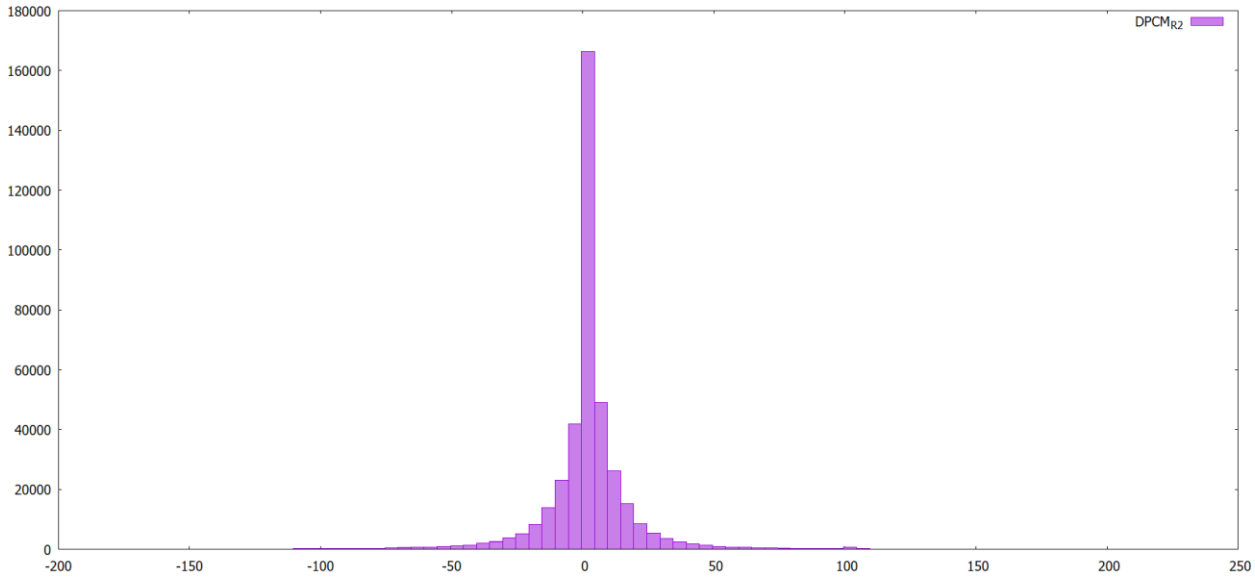

Рисунок 36. Гистограмма компоненты R, DPCM по 2 правилу

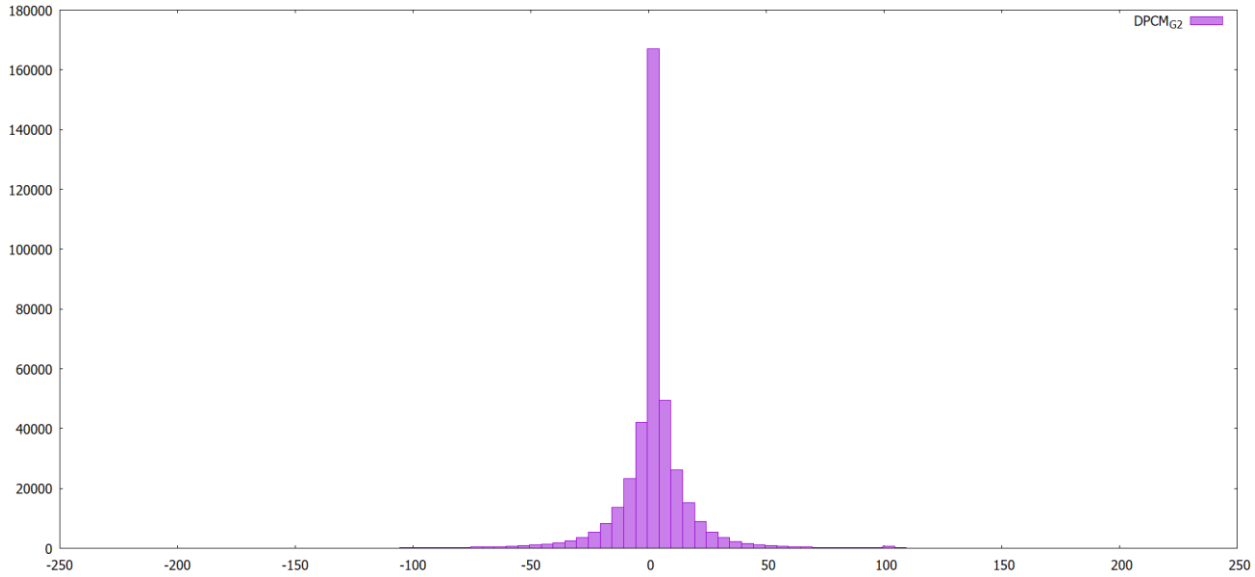

Рисунок 37. Гистограмма компоненты G, DPCM по 2 правилу

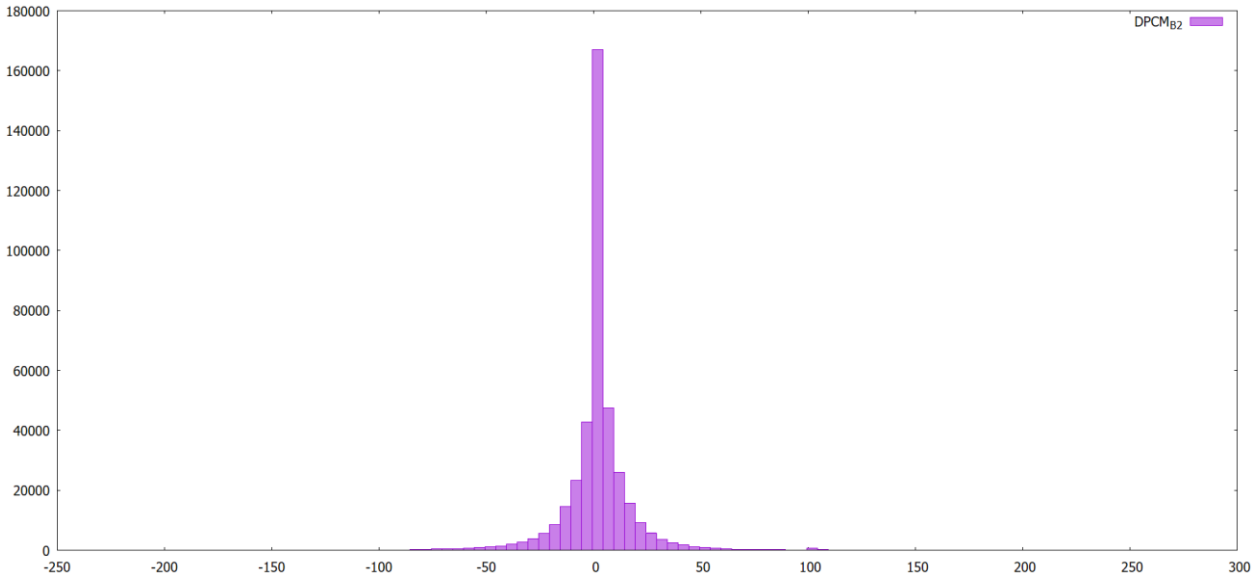

Рисунок 38. Гистограмма компоненты B, DPCM по 2 правилу

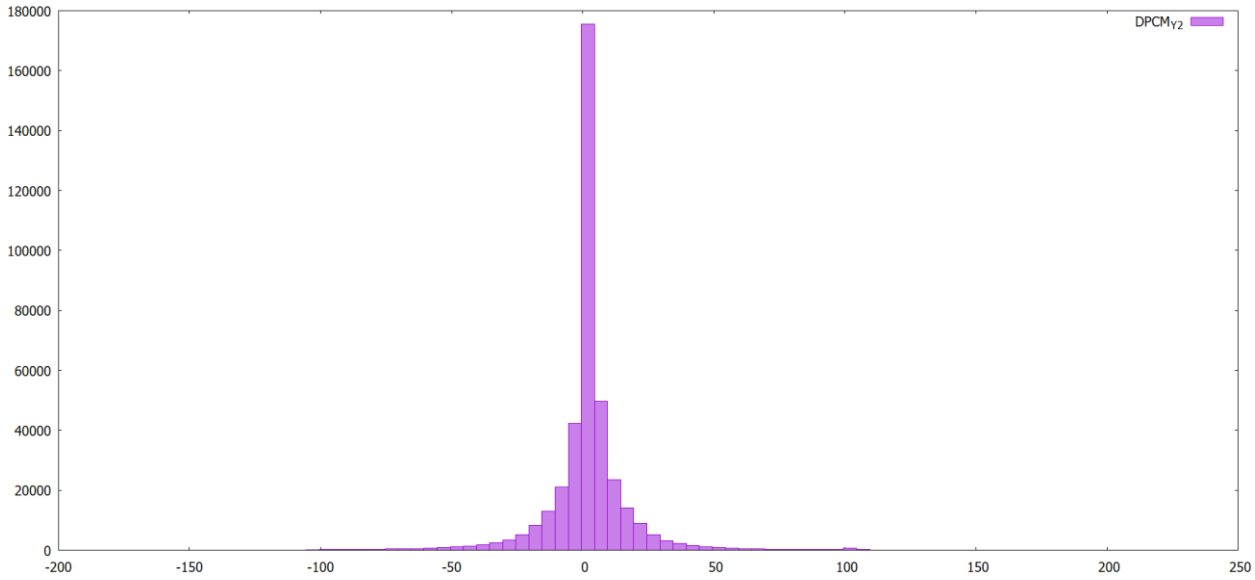

Рисунок 39. Гистограмма компоненты Y, DPCM по 2 правилу

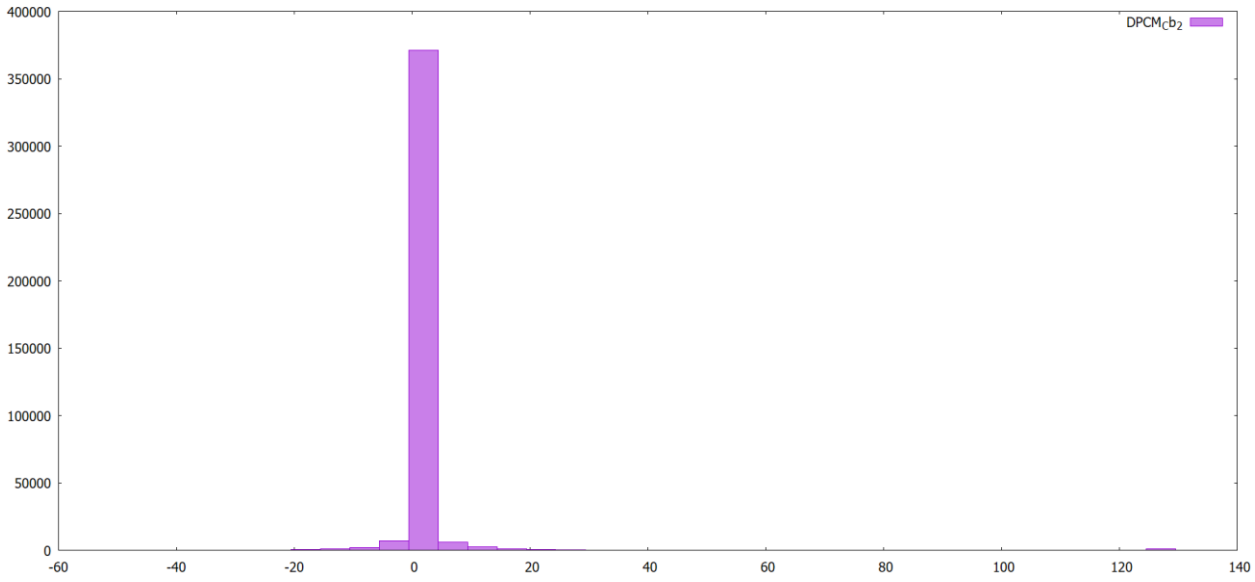

Рисунок 40. Гистограмма компоненты Cb, DPCM по 2 правилу

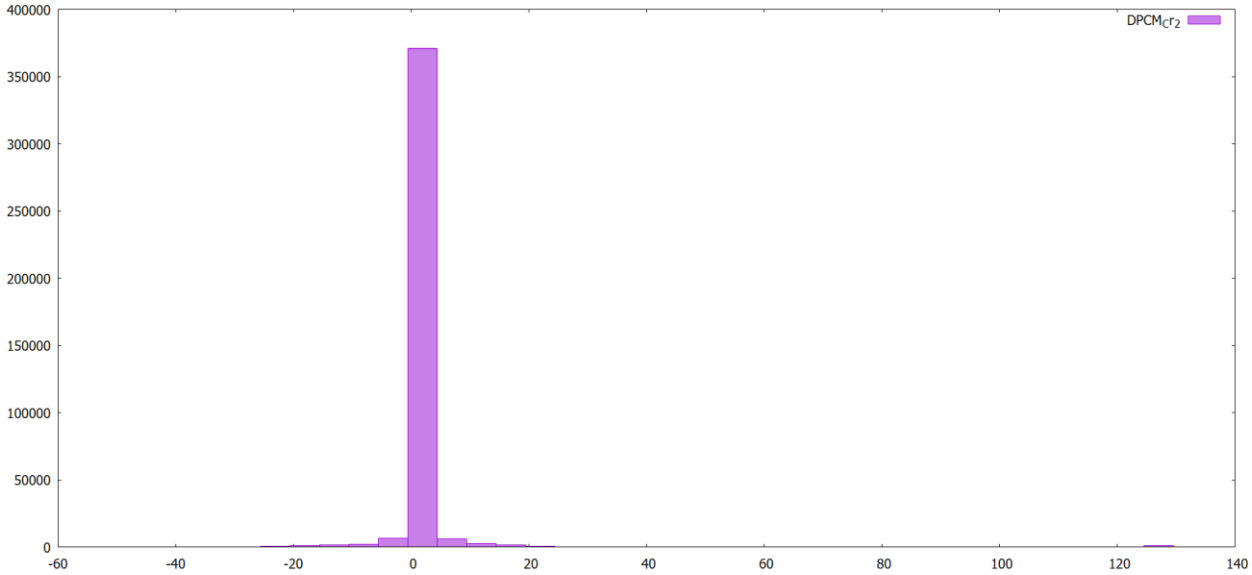

Рисунок 41. Гистограмма компоненты Cr, DPCM по 2 правилу

3) Массив DPCM, построенный по правилу 3

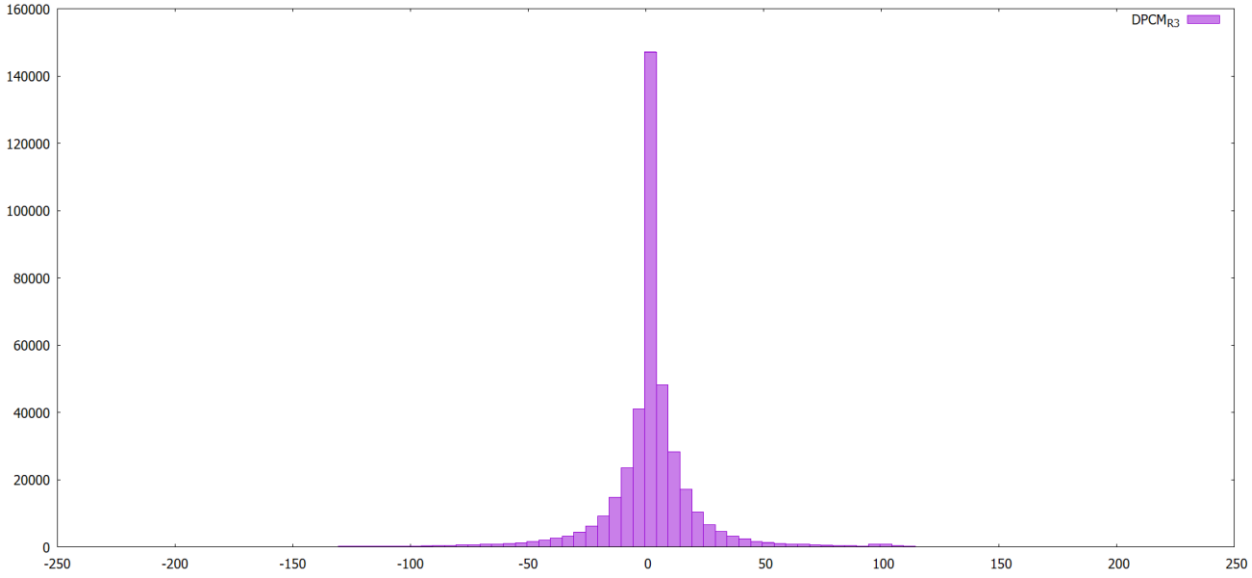

Рисунок 42. Гистограмма компоненты R, DPCM по 3 правилу

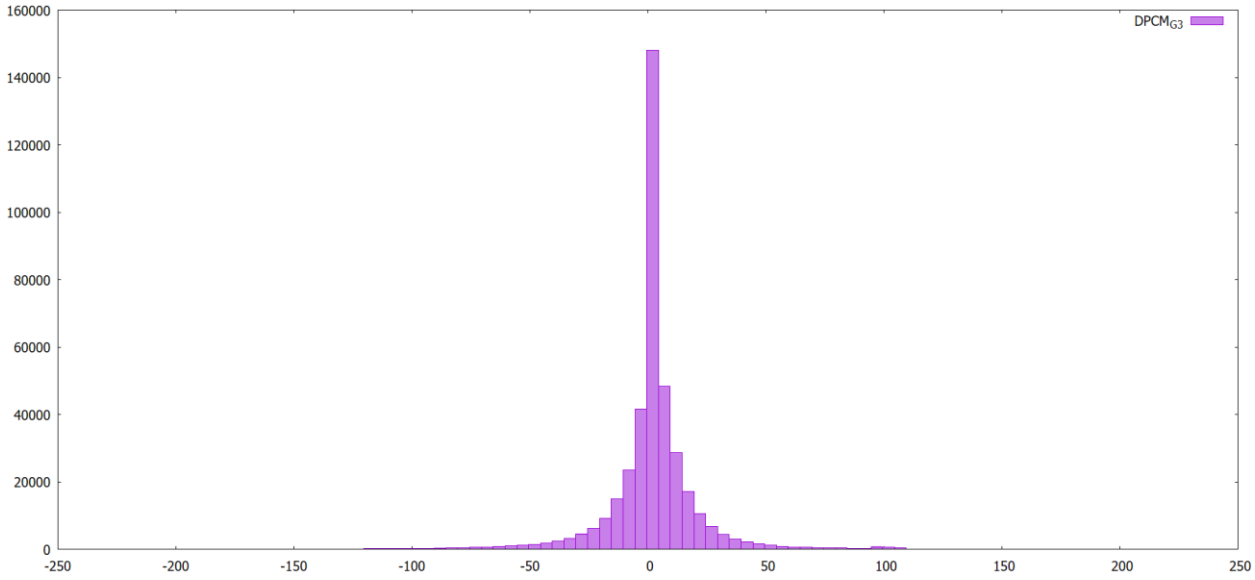

Рисунок 43. Гистограмма компоненты G, DPCM по 3 правилу

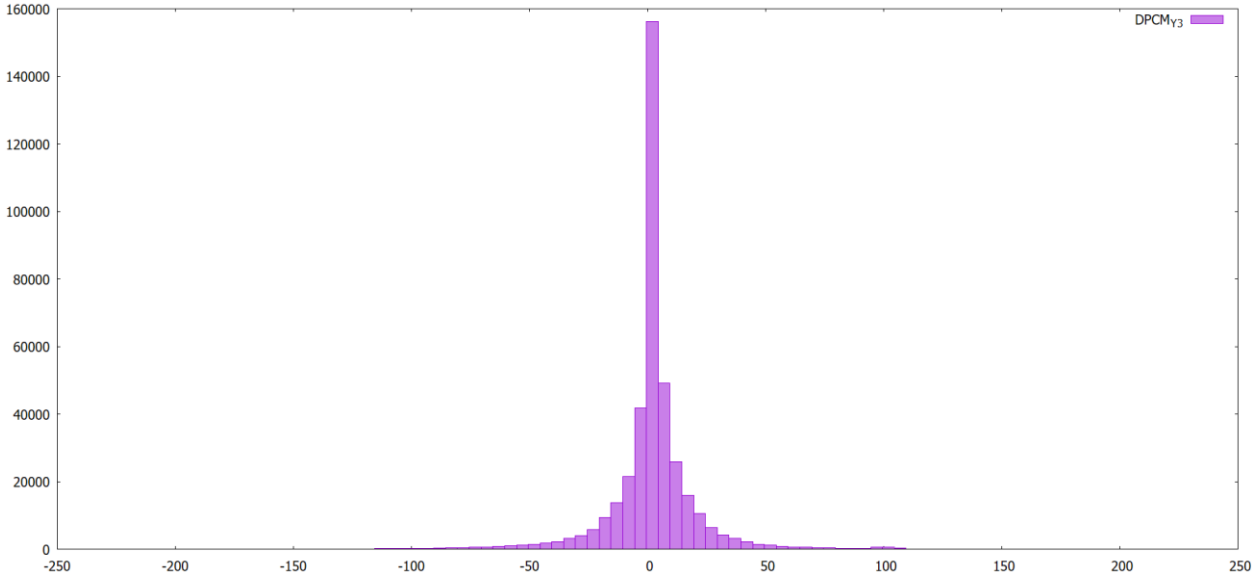

Рисунок 44. Гистограмма компоненты Y, DPCM по 3 правилу

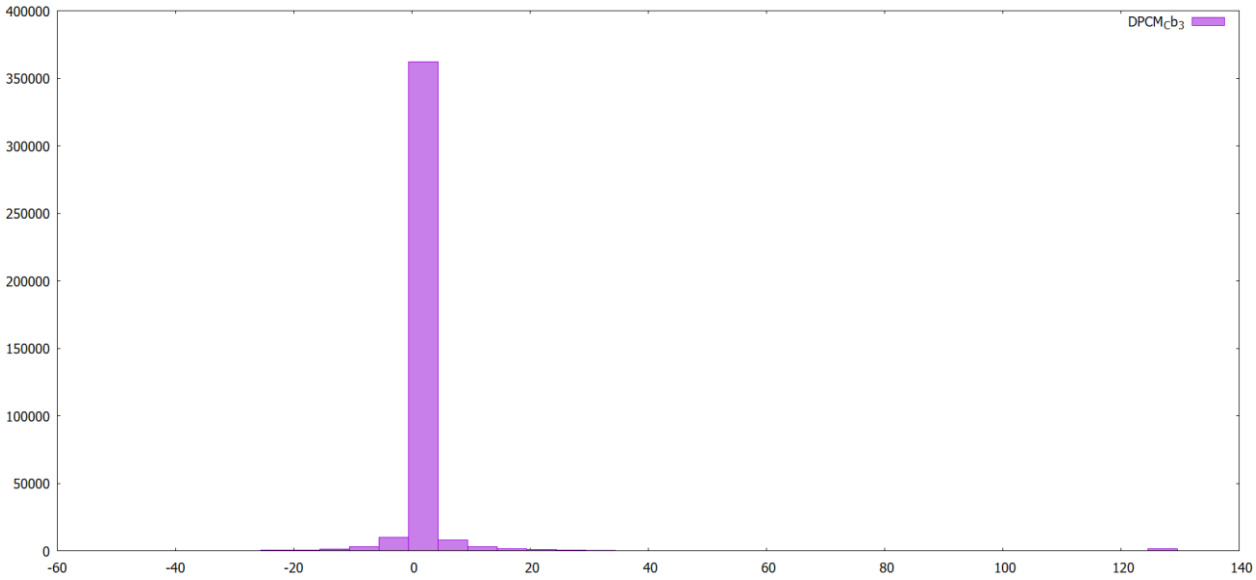

Рисунок 45. Гистограмма компоненты Cb, DPCM по 3 правилу

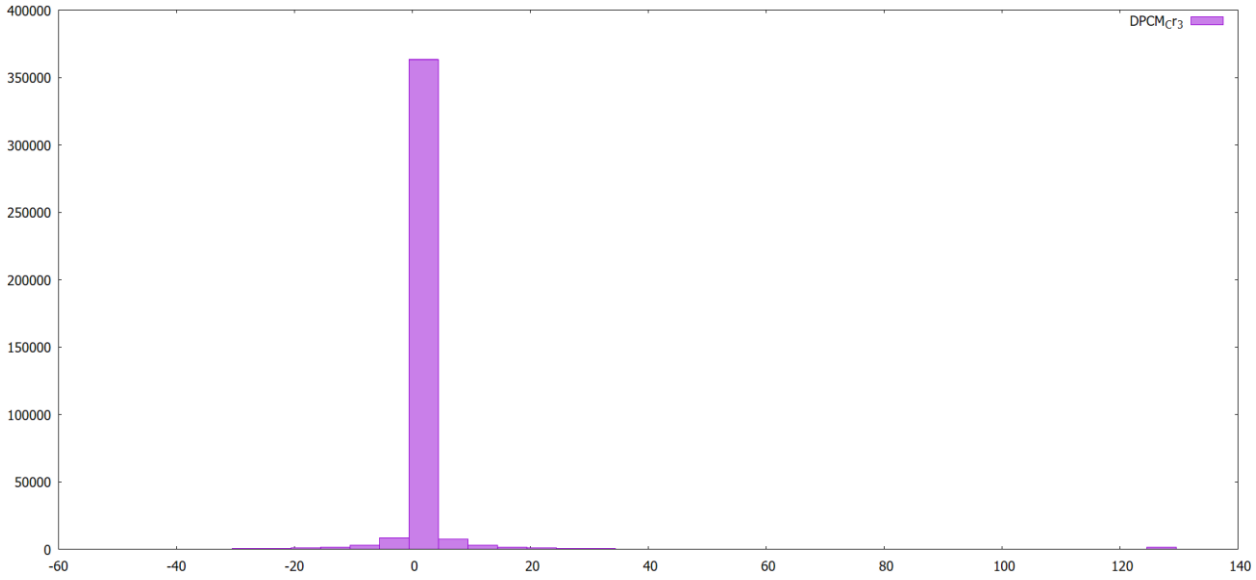

Рисунок 46. Гистограмма компоненты Cr, DPCM по 3 правилу

4) Массив DPCM, построенный по правилу 4

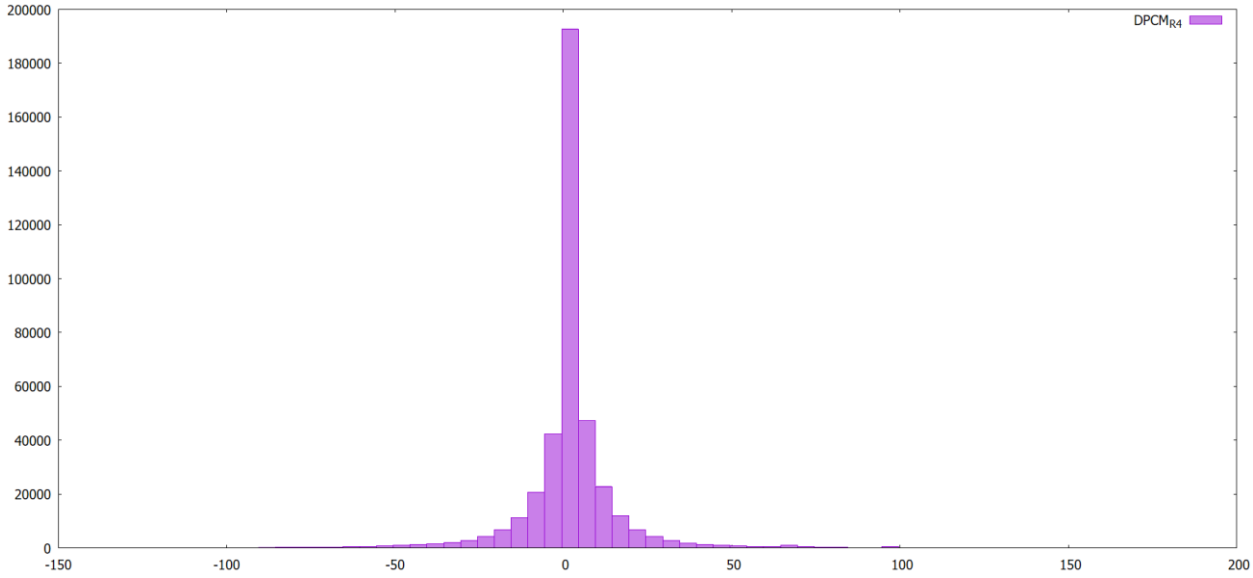

Рисунок 47. Гистограмма компоненты R, DPCM по 4 правилу

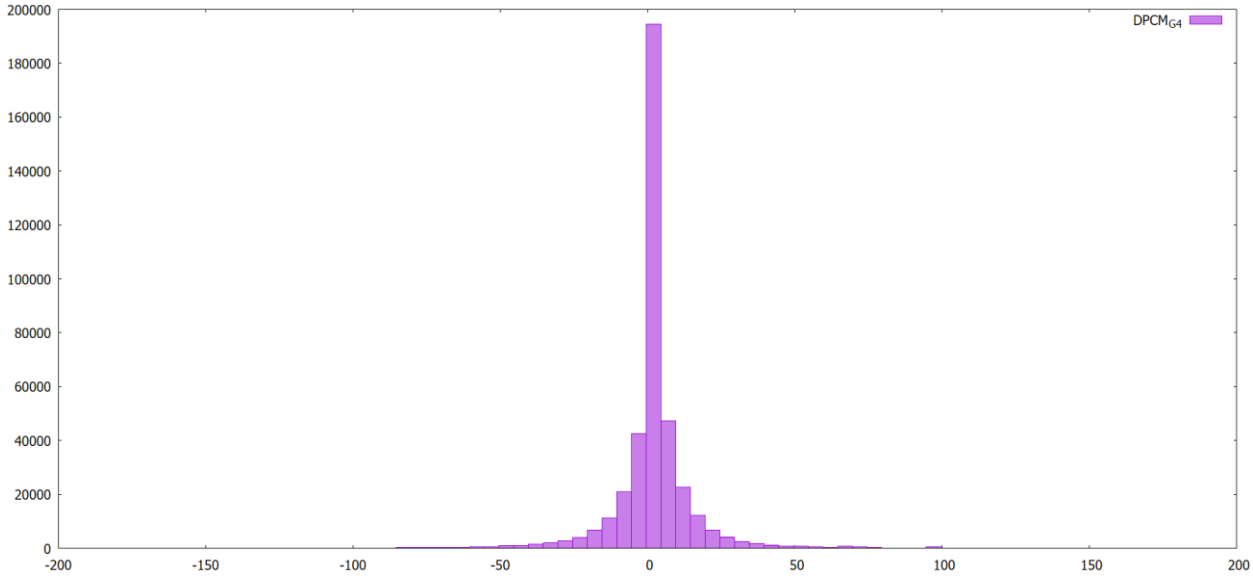

Рисунок 48. Гистограмма компоненты G, DPCM по 4 правилу

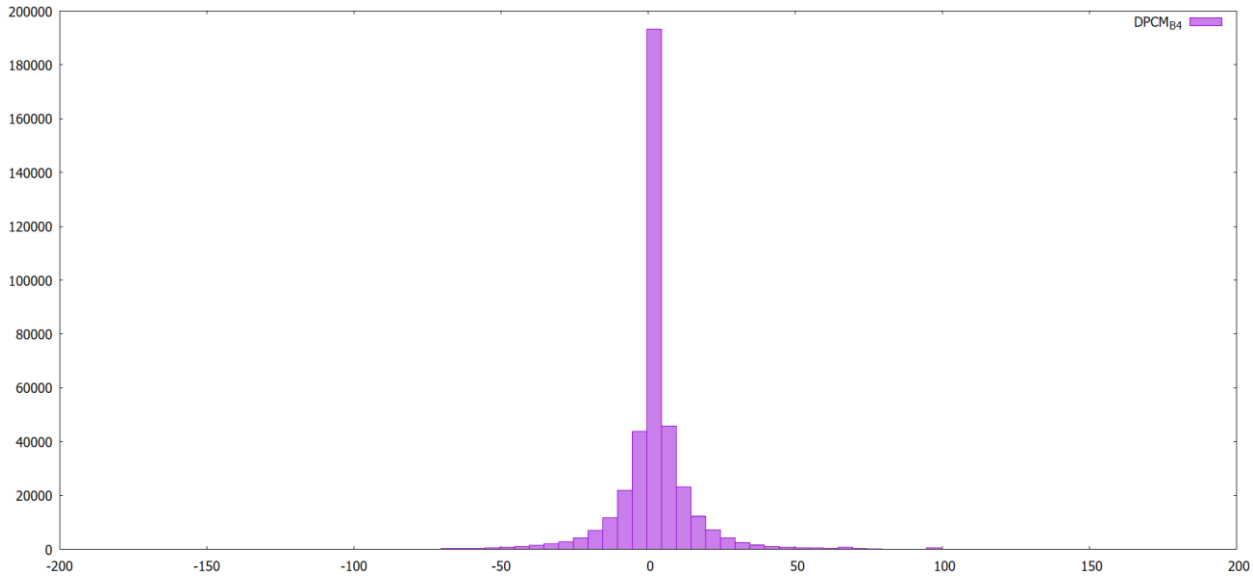

Рисунок 49. Гистограмма компоненты B, DPCM по 4 правилу

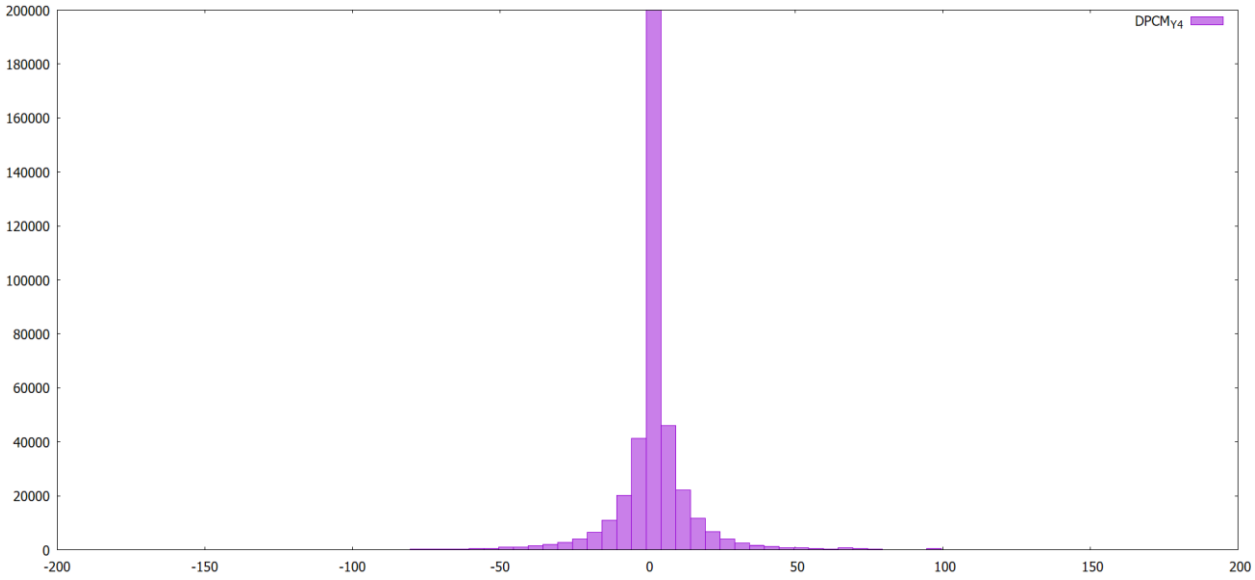

Рисунок 50. Гистограмма компоненты Y, DPCM по 4 правилу

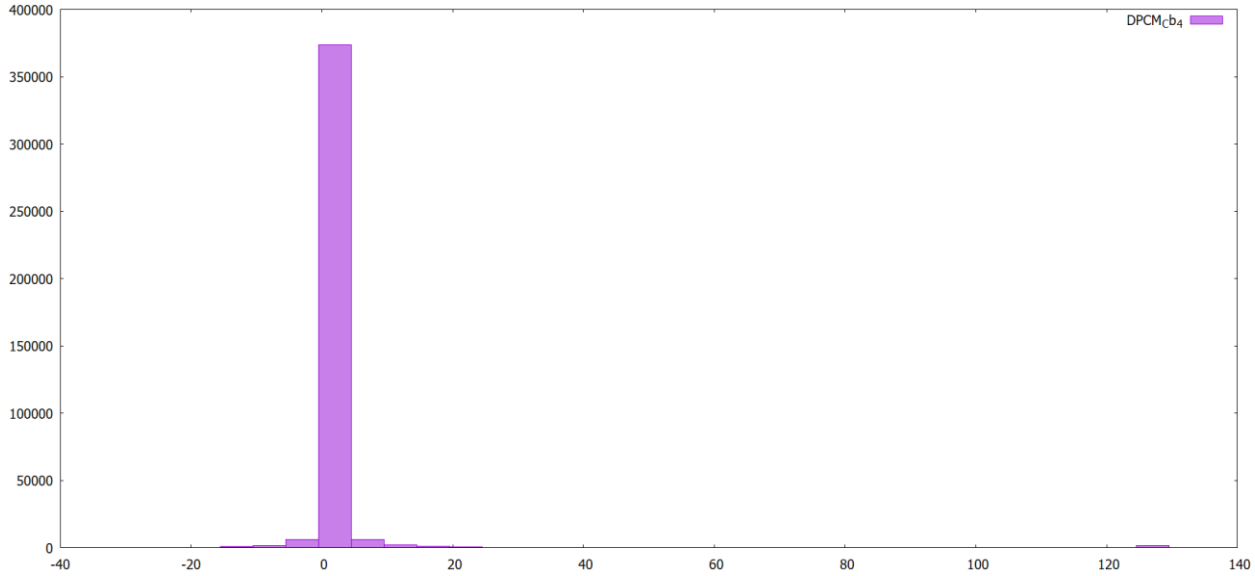

Рисунок 51. Гистограмма компоненты Cb, DPCM по 4 правилу

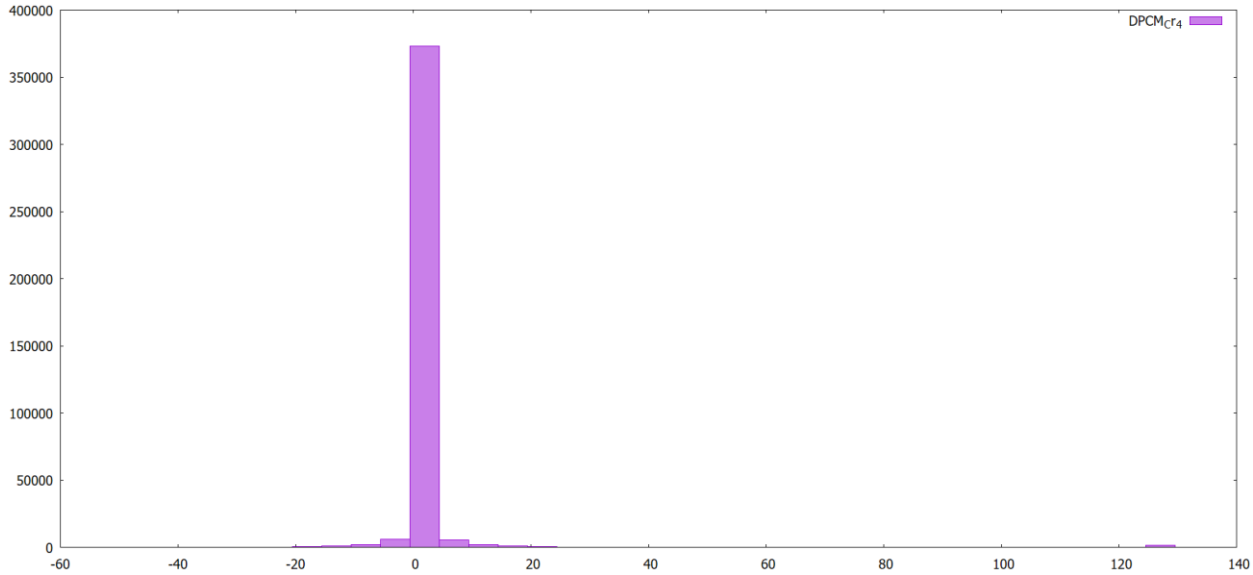

Рисунок 52. Гистограмма компоненты Cr, DPCM по 4 правилу

По результатам гистограмм видно, что все компоненты стремятся локализоваться в одной области, тем не менее, компоненты Cb и Cr, опять же более сосредоточены в одной области, по сравнению с остальными компонентами.

16. Оценка энтропии DPCM

```
H_DPCM_1_R = 5.39191
H_DPCM_1_G = 5.3608
H_DPCM_1_B = 5.34612
H_DPCM_1_Y = 5.21826
H_DPCM_1_Cb = 1.84961
H_DPCM_1_Cr = 1.74157
```

Рисунок 53. Оценка энтропии компонент, DPCM по 1 правилу

```
H_DPCM_2_R = 5.78261
H_DPCM_2_G = 5.75428
H_DPCM_2_B = 5.74346
H_DPCM_2_Y = 5.62962
H_DPCM_2_Cb = 2.13703
H_DPCM_2_Cr = 1.9967
```

Рисунок 54. Оценка энтропии компонент, DPCM по 2 правилу

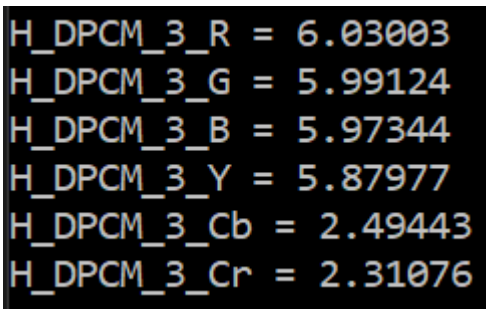

Рисунок 55. Оценка энтропии компонент, DPCM по 3 правилу

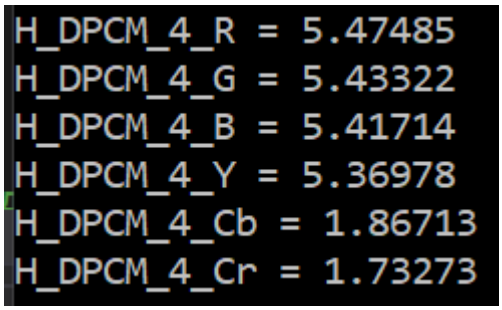

Рисунок 56. Оценка энтропии компонент, DPCM по 4 правилу

Как видно по результатам работы программы, энтропия значительно снизилась. Лучших результатов энтропии удалось добиться при работе алгоритма DPCM по первому и четвертому правилу.

17. Индивидуальное задание.

1) Изменение местоположения пикселей исходного файла для получения следующих результатов:

b) результирующий файл должен содержать исходное изображение, отраженное относительно горизонтали (эффект отражения в воде);

Чтобы отразить файл по горизонтали, необходимо отзеркалить порядок строк, то есть поменять первую с последней и так далее.

Рисунок 57. Отраженное по горизонтали изображение

Выводы

Была изучена структура BMP-изображения формата RGB24: были изучены заголовки, что помогло в дальнейшем извлечь такую важную информацию, как ширина и высота изображения.

Были исследованы корреляционные свойства компонент R, G, B, выявлено, что зависимость между ними достаточно высокая, особенно между компонентами R и G.

Для того, чтобы избежать высокой корреляции используется формат YCbCr, так как в данном формате меньшая корреляционная связь между компонентами, что так же позволяет обеспечить лучшее сжатие. Но формат YCbCr обладает и минусами: при восстановлении исходного изображения теряется часть информации. Для оценки потери информации и качества восстановления используется PSNR (пиковое отношение сигнала к шуму), так значения PSNR возрастают при восстановлении исходного изображения из формата YCbCr.

Децимация (частичное исключение информации) позволяет увеличить степень сжатия (для оценки используется оценка энтропии), но теряется качество изображения. Так, в данной программной реализации самым лучшим вариантом децимации оказалась децимация в 2 раза путем использования среднего арифметического, а самым худшим вариантом – децимация в 4 раза путем исключения строк и столбцов.

Анализ эффективности разностного кодирования показал, что он улучшает показатели кодирования и сжатия, в данной программной реализации лучшей

эффективности удалось достичь при использовании первого и четвертого правила.

Листинг программы:

```
#define _CRT_SECURE_NO_WARNINGS
#define _USE_MATH_DEFINES
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

typedef struct BFH
{
	short bfType;
	int bfSize;
	short bfReserved1;
	short bfOffBits;;
	int bfReserved2;
} MBITMAPFILEHEADER;
typedef struct BIH
{
	int biSize;
	int biWidth;
	int biHeight;
	short int biPlanes;
	short int biBitCount;
	int biCompression;
	int biSizeImage;
	int biXPelsPerMeter;
	int biYPelsPerMeter;
	int biClrUsed;
	int biClrImportant;
} MBITMAPINFOHEADER;
typedef struct RGB
{
	unsigned char rgbBlue;
	unsigned char rgbGreen;
	unsigned char rgbRed;
}MRGBQUAD;

unsigned char min_R = 0;;
unsigned char min_G = 0;
unsigned char min_B = 0;
unsigned char max_R = 0;
unsigned char max_G = 0;
unsigned char max_B = 0;

MRGBQUAD** loading_bmp(FILE* f, MBITMAPFILEHEADER* bfh,
MBITMAPINFOHEADER* bih)
{
	int k = 0;
```

```
	k = fread(bfh, sizeof(*bfh) - 2, 1, f);	//читаем файловый заголовок в структуру BITMAPFILEHEADER
	if (k == 0)
	{
		std::cout << "reading error";
		return 0;
	}

	k = fread(bih, sizeof(*bih), 1, f);	//Читаем заголовок изображения в структуру BITMAPINFOHEADER
	if (k == NULL)
	{
		std::cout << "reading error";
		return 0;
	}

	int a = abs(bih->biHeight);
	int b = abs(bih->biWidth);
	MRGBQUAD** rgb = new MRGBQUAD * [a];
	for (int i = 0; i < a; i++)
	{
		rgb[i] = new MRGBQUAD[b];
	}
	int pad = 4 - (b * 3) % 4;
	for (int i = 0; i < a; i++)
	{
		fread(rgb[i], sizeof(MRGBQUAD), b, f);
		if (pad != 4)
		{
			fseek(f, pad, SEEK_CUR);
		}
	}
	return rgb;
}

MRGBQUAD** getR(MRGBQUAD** rgb1, int a, int b) {
	MRGBQUAD** g = new MRGBQUAD * [a];
	for (int i = 0; i < a; i++)
	{
		g[i] = new MRGBQUAD[b];
	}
	for (int i = 0; i < a; i++) {
		for (int j = 0; j < b; j++) {
			g[i][j].rgbGreen = 0;
			g[i][j].rgbBlue = 0;
			g[i][j].rgbRed = rgb1[i][j].rgbRed;
		}
	}
	return g;
}
MRGBQUAD** getG(MRGBQUAD** rgb1, int a, int b) {
	MRGBQUAD** g = new MRGBQUAD * [a];
```

```
	for (int i = 0; i < a; i++)
	{
		g[i] = new MRGBQUAD[b];
	}
	for (int i = 0; i < a; i++) {
		for (int j = 0; j < b; j++) {
			g[i][j].rgbGreen = rgb1[i][j].rgbGreen;
			g[i][j].rgbBlue = 0;
			g[i][j].rgbRed = 0;
		}
	}
	return g;
}
MRGBQUAD** getB(MRGBQUAD** rgb1, int a, int b) {
	MRGBQUAD** g = new MRGBQUAD * [a];
	for (int i = 0; i < a; i++)
	{
		g[i] = new MRGBQUAD[b];
	}
	for (int i = 0; i < a; i++) {
		for (int j = 0; j < b; j++) {
			g[i][j].rgbGreen = 0;
			g[i][j].rgbBlue = rgb1[i][j].rgbBlue;
			g[i][j].rgbRed = 0;
		}
	}
	return g;
}
void write_bmp(FILE* f, MRGBQUAD** rgbb, MBITMAPFILEHEADER* bfh,
MBITMAPINFOHEADER* bih, int a, int b)
{
	bih->biHeight = a;
	bih->biWidth = b;
	fwrite(bfh, sizeof(*bfh) - 2, 1, f);
	fwrite(bih, sizeof(*bih), 1, f);
	int pad = 4 - ((b) * 3) % 4;
	char buf = 0;
	for (int i = 0; i < a; i++)
	{
		fwrite((rgbb[i]), sizeof(MRGBQUAD), b, f);
		if (pad != 4)
		{
			fwrite(&buf, 1, pad, f);
		}
	}
}

double Get_math_ozh(vector<vector<double>> rgb, int h, int w) {
	double res = 0;
	double WH = (double)w * (double)h;

	for (int i = 0; i < rgb.size(); i++) {
```

```
		for (int j = 0; j < rgb[0].size(); j++) {
			if (rgb[i].size() != 0)
				res += rgb[i][j];
		}
	}
	res = res / WH;
	return res;
}
double Get_dispersion(vector<vector<double>> rgb, int h, int w) {
	double res = 0;
	double WH = (double)w * (double)h;
	double m = Get_math_ozh(rgb, h, w);

	for (int i = 0; i < rgb.size(); i++) {
		for (int j = 0; j < rgb[0].size(); j++) {
			if (rgb[i].size() != 0)
				res += pow((rgb[i][j] - m), 2);
		}
	}
	res = res / (WH - 1);
	return sqrt(res);
}
double Correlation_func(vector<vector<double>> A, vector<vector<double>> B, int h, int w) {
	double m1 = Get_math_ozh(A, h, w);
	double m2 = Get_math_ozh(B, h, w);

	double d1 = Get_dispersion(A, h, w);
	double d2 = Get_dispersion(B, h, w);

	for (int i = 0; i < A.size(); i++) {
		for (int j = 0; j < A[0].size(); j++) {

			if (A[i].size() != 0 && B[i].size() != 0) {
				A[i][j] = A[i][j] - m1;
				B[i][j] = B[i][j] - m2;
				A[i][j] = A[i][j] * B[i][j];
			}
		}
	}
	double res = Get_math_ozh(A, h, w) / (d1 * d2);
	return res;
}
vector<vector<double>> calculateFirstSelection(vector<vector<double>>& array, int x, int y) {
	/*int height = array[0].size();
	int width = array.size();*/
	int width = array.size();
	int height = array[0].size();
	vector<vector<double>> result(width);

	for (int i = 0; i < width; ++i) {
		result[i].resize(height);
	}
```

```
	if (y >= 0) {
		if (x >= 0) {
			for (int j = 0; j < width - x; ++j) {
				for (int i = 0; i < height - y; ++i) {
					result[j][i] = array[j][i];
				}
			}
		}
		else {
			for (int j = -x; j < width; ++j) {
				for (int i = 0; i < height - y; ++i) {
					result[j + x][i] = array[j][i];
				}
			}
		}
	}
	else {
		if (x >= 0) {
			for (int j = 0; j < width - x; ++j) {
				for (int i = -y; i < height; ++i) {
					result[j][i + y] = array[j][i];
				}
			}
		}
		else {
			for (int j = -x; j < width; j++) {
				for (int i = -y; i < height; i++) {
					result[j + x][i + y] = array[j][i];
				}
			}
		}
	}
	return result;
}
vector<vector<double>> calculateSecondSelection(vector<vector<double>>& array, int x, int y)
{
	int height = array[0].size();
	int width = array.size();
	vector<vector<double>> result(width);

	for (int i = 0; i < width; ++i) {
		result[i].resize(height);
	}
	if (y >= 0) {
		if (x >= 0) {
			for (int j = x; j < width; ++j) {
				for (int i = y; i < height; ++i) {
					result[j - x][i - y] = array[j][i];
				}
			}
		}
```

```
            else {
                for (int j = 0; j < width + x; ++j) {
                    for (int i = y; i < height; ++i) {
                        result[j][i - y] = array[j][i];
                    }
                }
            }
        }
        else {
            if (x >= 0) {
                for (int j = x; j < width; ++j) {
                    for (int i = 0; i < height + y; ++i) {
                        result[j - x][i] = array[j][i];
                    }
                }
            }
            else {
                for (int j = 0; j < width + x; ++j) {
                    for (int i = 0; i < height + y; ++i) {
                        result[j][i] = array[j][i];
                    }
                }
            }
        }
        return result;
}
void write_txt(string filename, vector<double> result, int width) {
        ofstream fout(filename);
        int count = 0;
        if (fout.is_open()) {
            int j = 0;
            for (int i = -width / 4; i < width / 4; i += 2) {
                fout << i << " " << result[j] << "\n";
                j++;
            }
        }
        else
            cout << "Can`t open the file" << endl;
        fout.close();
}
vector<double> Autocorrelation_func(vector<vector<double>>& color, int y) {
        int width = color.size();
        int height = color[0].size();
        vector<double> array(width / 4);

        for (int x = -width / 4, counter = 0; x < width / 4; x += 2, counter++) {
            vector<vector<double>> firstSelection = calculateFirstSelection(color, x, y);
            vector<vector<double>> secondSelection = calculateSecondSelection(color, x, y);
            array[counter] = Correlation_func(firstSelection, secondSelection, height, width);
        }
        return array;
}
```

```
MRGBQUAD** Grayscale(vector<vector<double>> Y, int a, int b) {
	MRGBQUAD** res = new MRGBQUAD * [a];
	for (int i = 0; i < a; i++) {
		res[i] = new MRGBQUAD[b];
	}

	for (int i = 0; i < a; i++) {
		for (int j = 0; j < b; j++) {
			res[i][j].rgbGreen = Y[i][j];
			res[i][j].rgbBlue = Y[i][j];
			res[i][j].rgbRed = Y[i][j];
		}
	}
	return res;
}

unsigned char saturation(double x, int xmin, int xmax) {
	if (x < (double)xmin)
		x = (double)xmin;
	if (x > (double)xmax)
		x = (double)xmax;
	return (unsigned char)x;
}
MRGBQUAD** getRGBfromYreverse(vector<vector<double>> Y, vector<vector<double>>
Cb, vector<std::vector<double>> Cr, int a, int b) {
	MRGBQUAD** g = new MRGBQUAD * [a];
	for (int i = 0; i < a; i++)
	{
		g[i] = new MRGBQUAD[b];
	}
	for (int i = 0; i < a; i++) {
		for (int j = 0; j < b; j++) {
			g[i][j].rgbGreen = saturation(Y[i][j] - 0.714 * (Cr[i][j] - 128.0) - 0.334 *
(Cb[i][j] - 128.0), min_G, max_G);
			g[i][j].rgbRed = saturation(Y[i][j] + 1.402 * (Cr[i][j] - 128.0), min_R,
max_R);
			g[i][j].rgbBlue = saturation(Y[i][j] + 1.772 * (Cb[i][j] - 128.0), min_B,
max_B);
		}
	}
	return g;
}
double sum_sqr_differense(vector<vector<double>> v1, vector<vector<double>> v2) {
	double res = 0;
	for (int i = 0; i < v1.size(); i++) {
		for (int j = 0; j < v1[0].size(); j++) {
			res += pow((v1[i][j] - v2[i][j]), 2);
		}
	}
	return res;
}
```

```
double PSNR(vector<vector<double>> v1, vector<vector<double>> v2) {
	double PSNR = 10 * log10(v1.size() * v1[0].size() * pow((pow(2, 8) - 1), 2) /
sum_sqr_differense(v1, v2));
	return PSNR;
}
void find_min_max(MRGBQUAD** rgb, int h1, int w1) {
	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {
			if (rgb[i][j].rgbRed > max_R)
				max_R = rgb[i][j].rgbRed;
			if (rgb[i][j].rgbGreen > max_G)
				max_G = rgb[i][j].rgbGreen;
			if (rgb[i][j].rgbBlue > max_B)
				max_B = rgb[i][j].rgbBlue;
			if (rgb[i][j].rgbRed < min_R)
				min_R = rgb[i][j].rgbRed;
			if (rgb[i][j].rgbGreen < min_G)
				min_G = rgb[i][j].rgbGreen;
			if (rgb[i][j].rgbBlue < min_B)
				min_B = rgb[i][j].rgbBlue;
		}
	}
}

std::vector<vector<double>> decimation_even_exception(vector<vector<double>> Cb, int h1,
int w1, int n) {
	std::vector<std::vector<double>> newCb(h1 / n);
	int count = 0;
	for (int i = 0; i < h1; i += n) {
		for (int j = 0; j < w1; j += n) {
			newCb[count].push_back(Cb[i][j]);
		}
		count++;
	}
	//restore
	std::vector<std::vector<double>> restCb(h1);
	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {
			restCb[i].push_back(newCb[i / n][j / n]);
		}
	}
	return restCb;
}
double Arithmetic_mean(std::vector<std::vector<double>>& Cb, int i, int j, int n) {
	double sum = 0;
	for (int k = 0; k < n; k++) {
		for (int t = 0; t < n; t++) {
			sum += Cb[i + k][j + t];
		}
	}
	sum = sum / (double)(n * n);
	return sum;
```

```
}
std::vector<vector<double>> decimation_by_arithmetic_mean(vector<vector<double>>& Cb,
int h1, int w1, int n) {
	std::vector<std::vector<double>> newCb(h1 / n);
	int count = 0;
	for (int i = 0; i < h1; i += n) {
		for (int j = 0; j < w1; j += n) {
			newCb[count].push_back(Arithmetic_mean(Cb, i, j, n));
		}
		count++;
	}
	//restore
	std::vector<std::vector<double>> restCb(h1);
	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {
			restCb[i].push_back(newCb[i / n][j / n]);
		}
	}
	return restCb;
}
double get_PSNR_only_R(MRGBQUAD** restoredRGB, vector<vector<double>>& R, int h1,
int w1) {
	std::vector<std::vector<double>> restR(h1);
	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {
			restR[i].push_back(restoredRGB[i][j].rgbRed);
		}
	}
	return PSNR(R, restR);

}
double get_PSNR_only_G(MRGBQUAD** restoredRGB, vector<vector<double>>& G, int h1,
int w1) {
	std::vector<std::vector<double>> restG(h1);
	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {
			restG[i].push_back(restoredRGB[i][j].rgbGreen);
		}
	}
	return PSNR(G, restG);
}
double get_PSNR_only_B(MRGBQUAD** restoredRGB, vector<vector<double>>& B, int h1,
int w1) {

	std::vector<std::vector<double>> restB(h1);
	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {

			restB[i].push_back(restoredRGB[i][j].rgbBlue);
		}
	}
	return PSNR(B, restB);
```

```
}

void file_write(const char* s, std::vector<std::vector<double>>& v, int h1, int w1) {
	std::ofstream fout1;
	fout1.open(s);
	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {
			fout1 << v[i][j] << "\n";
		}
	}
	fout1.close();
}

double entropy(std::vector<std::vector<double>> rgb) {
	double H = 0.0;
	std::vector<double> p;
	for (int i = 0; i < 256; i++) {
		p.push_back(0);
	}
	for (int i = 0; i < rgb.size(); i++) {
		for (int j = 0; j < rgb[0].size(); j++) {
			int tmp = rgb[i][j];
			tmp += 256;
			tmp = tmp % 256;
			p[tmp]++;
		}
	}
	for (int i = 0; i < 256; i++) {
		p[i] = p[i] / (double)(rgb.size() * rgb[0].size());
	}

	for (int i = 0; i < 256; i++) {
		if (p[i] != 0)
			H += (double)(p[i] * std::log2(p[i]));
	}
	H = (double)(-H);
	return H;
}

std::vector<std::vector<double>> DPCM1(std::vector<std::vector<double>>& v) {
	std::vector<std::vector<double>> res(v.size());
	for (int i = 0; i < v.size(); i++) {
		for (int j = 0; j < v[0].size(); j++) {
			if (j == 0 || i == 0) {
				res[i].push_back(v[i][j]);
				continue;
			}
			res[i].push_back((v[i][j] - v[i][j - 1]));

		}
	}
	return res;
```

```
}
std::vector<std::vector<double>> DPCM2(std::vector<std::vector<double>>& v) {
	std::vector<std::vector<double>> res(v.size());
	for (int i = 0; i < v.size(); i++) {
		for (int j = 0; j < v[0].size(); j++) {
			if (i == 0) {
				res[i].push_back(v[i][j]);
				continue;
			}
			res[i].push_back(v[i][j] - v[i - 1][j]);

		}
	}
	return res;
}
std::vector<std::vector<double>> DPCM3(std::vector<std::vector<double>>& v) {
	std::vector<std::vector<double>> res(v.size());
	for (int i = 0; i < v.size(); i++) {
		for (int j = 0; j < v[0].size(); j++) {
			if (i == 0 || j == 0) {
				res[i].push_back(v[i][j]);
				continue;
			}
			res[i].push_back(v[i][j] - v[i - 1][j - 1]);

		}
	}
	return res;
}
std::vector<std::vector<double>> DPCM4(std::vector<std::vector<double>>& v) {
	std::vector<std::vector<double>> res(v.size());
	for (int i = 0; i < v.size(); i++) {
		for (int j = 0; j < v[0].size(); j++) {
			if (i == 0 || j == 0) {
				res[i].push_back(v[i][j]);
				continue;
			}
			res[i].push_back(v[i][j] - (v[i - 1][j] + v[i][j - 1] + v[i - 1][j - 1]) / 3);

		}
	}
	return res;
}

MRGBQUAD** horizonral_reflection(MRGBQUAD** rgb1, int a, int b) {
	MRGBQUAD** g = new MRGBQUAD * [a];
	for (int i = 0; i < a; i++)
	{
		g[i] = new MRGBQUAD[b];
	}
	for (int i = 0; i < a; i++) {
		for (int j = 0; j < b; j++) {
```

```
                        g[i][j].rgbRed = rgb1[a - i - 1][j].rgbRed;
                        g[i][j].rgbGreen = rgb1[a - i - 1][j].rgbGreen;
                        g[i][j].rgbBlue = rgb1[a - i - 1][j].rgbBlue;
                }
        }
        return g;
}

int main()
{

//№2
        cout << "#2" << endl;
        MBITMAPFILEHEADER bfh;
        MBITMAPINFOHEADER bih;
        FILE* f1;
        f1 = fopen("BMP.bmp", "rb");
        if (f1 == NULL)
        {
                std::cout << "reading error";
                return 0;
        }
        MRGBQUAD** rgb1 = loading_bmp(f1, &bfh, &bih);
        fclose(f1);
        /*std::cout << "biBitCount = " << bih.biBitCount;
        std::cout << std::endl;*/
//№3
        cout << "#3" << endl;
        int h1 = abs(bih.biHeight);
        int w1 = abs(bih.biWidth);

        MRGBQUAD** green = getG(rgb1, h1, w1);
        std::vector<std::vector<double>> g(h1);
        for (int i = 0; i < h1; i++) {
                for (int j = 0; j < w1; j++) {
                        g[i].push_back((double)green[i][j].rgbGreen);
                }
        }
        FILE* fgreen = fopen("Files\\3\\Green.bmp", "wb");
        write_bmp(fgreen, green, &bfh, &bih, h1, w1);
        fclose(fgreen);

        MRGBQUAD** blue = getB(rgb1, h1, w1);
        std::vector<std::vector<double>> b(h1);
        for (int i = 0; i < h1; i++) {
                for (int j = 0; j < w1; j++) {
                        b[i].push_back((double)blue[i][j].rgbBlue);
                }
        }
        FILE* fblue = fopen("Files\\3\\Blue.bmp", "wb");
        write_bmp(fblue, blue, &bfh, &bih, h1, w1);
        fclose(fblue);
```

```
	MRGBQUAD** red = getR(rgb1, h1, w1);
	std::vector<std::vector<double>> r(h1);
	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {
			r[i].push_back((double)red[i][j].rgbRed);
		}
	}
	FILE* fred = fopen("Files\\3\\Red.bmp", "wb");
	write_bmp(fred, red, &bfh, &bih, h1, w1);
	fclose(fred);

//№4.а)
	cout << "#4 a)" << endl;
	cout << "correlation R-G = " << Correlation_func(r, g, h1, w1) << endl;
	cout << "correlation R-B = " << Correlation_func(r, b, h1, w1) << endl;
	cout << "correlation B-G = " << Correlation_func(b, g, h1, w1) << endl << endl;

//№4.б)
	/*
	//красный
	cout << "#4 b)" << endl;
	vector<double> autocorrRed1 = Autocorrelation_func(r, -10);
	write_txt("Files\\4\\autocorrRed-10.txt", autocorrRed1, h1);
	cout << "autocorr Red1 done!" << endl;

	vector<double> autocorrRed2 = Autocorrelation_func(r, -5);
	write_txt("Files\\4\\autocorrRed-5.txt", autocorrRed2, h1);
	cout << "autocorr Red2 done!" << endl;

	vector<double> autocorrRed3 = Autocorrelation_func(r, 0);
	write_txt("Files\\4\\autocorrRed0.txt", autocorrRed3, h1);
	cout << "autocorr Red3 done!" << endl;

	vector<double> autocorrRed4 = Autocorrelation_func(r, 5);
	write_txt("Files\\4\\autocorrRed5.txt", autocorrRed4, h1);
	cout << "autocorr Red4 done!" << endl;

	vector<double> autocorrRed5 = Autocorrelation_func(r, 10);
	write_txt("Files\\4\\autocorrRed10.txt", autocorrRed5, h1);
	cout << "autocorr Red5 done!" << endl;

	//зеленый
	vector<double> autocorrGreen1 = Autocorrelation_func(g, -10);
	write_txt("Files\\4\\autocorrGreen-10.txt", autocorrGreen1, h1);
	cout << "autocorr Green1 done!" << endl;

	vector<double> autocorrGreen2 = Autocorrelation_func(g, -5);
	write_txt("Files\\4\\autocorrGreen-5.txt", autocorrGreen2, h1);
	cout << "autocorr Green2 done!" << endl;

	vector<double> autocorrGreen3 = Autocorrelation_func(g, 0);
```

```
	write_txt("Files\\4\\autocorrGreen0.txt", autocorrGreen3, h1);
	cout << "autocorr Green3 done!" << endl;

	vector<double> autocorrGreen4 = Autocorrelation_func(g, 5);
	write_txt("Files\\4\\autocorrGreen5.txt", autocorrGreen4, h1);
	cout << "autocorr Green4 done!" << endl;

	vector<double> autocorrGreen5 = Autocorrelation_func(g, 10);
	write_txt("Files\\4\\autocorrGreen10.txt", autocorrGreen5, h1);
	cout << "autocorr Green5 done!" << endl;

	//голубой
	vector<double> autocorrBlue1 = Autocorrelation_func(b, -10);
	write_txt("Files\\4\\autocorrBlue-10.txt", autocorrBlue1, h1);
	cout << "autocorr Blue1 done!" << endl;

	vector<double> autocorrBlue2 = Autocorrelation_func(b, -5);
	write_txt("Files\\4\\autocorrBlue-5.txt", autocorrBlue2, h1);
	cout << "autocorr Blue2 done!" << endl;

	vector<double> autocorrBlue3 = Autocorrelation_func(b, 0);
	write_txt("Files\\4\\autocorrBlue0.txt", autocorrBlue3, h1);
	cout << "autocorr Blue3 done!" << endl;

	vector<double> autocorrBlue4 = Autocorrelation_func(b, 5);
	write_txt("Files\\4\\autocorrBlue5.txt", autocorrBlue4, h1);
	cout << "autocorr Blue4 done!" << endl;

	vector<double> autocorrBlue5 = Autocorrelation_func(b, 10);
	write_txt("Files\\4\\autocorrBlue10.txt", autocorrBlue5, h1);
	cout << "autocorr Blue5 done!" << endl;
	*/

//№5
	cout << "#5" << endl;
	vector<vector<double>> Y(h1);
	vector<vector<double>> Cb(h1);
	vector<vector<double>> Cr(h1);

	for (int i = 0; i < h1; i++) {
		for (int j = 0; j < w1; j++) {
			Y[i].push_back((double)rgb1[i][j].rgbRed * 0.299 +
(double)rgb1[i][j].rgbGreen * 0.587 + (double)rgb1[i][j].rgbBlue * 0.114);
			Cb[i].push_back(0.5643 * ((double)rgb1[i][j].rgbBlue - Y[i][j]) + 128);
			Cr[i].push_back(0.7132 * ((double)rgb1[i][j].rgbRed - Y[i][j]) + 128);
		}
	}

	cout << "correlation Y-Cb = " << Correlation_func(Y, Cb, h1, w1) << endl;
	cout << "correlation Y-Cr = " << Correlation_func(Y, Cr, h1, w1) << endl;
	cout << "correlation Cb-Cr = " << Correlation_func(Cb, Cr, h1, w1) << endl << endl;
```

```
//№6
	cout << "#6" << endl;
	MRGBQUAD** rgbY = Grayscale(Y, h1, w1);
	FILE* fY = fopen("Files\\6\\contentY.bmp", "wb");
	if (fY == NULL)
		cout << "Can`t open the image";
	write_bmp(fY, rgbY, &bfh, &bih, h1, w1);
	fclose(fY);

	MRGBQUAD** rgbCb = Grayscale(Cb, h1, w1);
	FILE* fCb = fopen("Files\\6\\contentCb.bmp", "wb");
	if (fCb == NULL)
		cout << "Can`t open the image";
	write_bmp(fCb, rgbCb, &bfh, &bih, h1, w1);
	fclose(fCb);

	MRGBQUAD** rgbCr = Grayscale(Cr, h1, w1);
	FILE* fCr = fopen("Files\\6\\contentCr.bmp", "wb");
	if (fCr == NULL)
		cout << "Can`t open the image";
	write_bmp(fCr, rgbCr, &bfh, &bih, h1, w1);
	fclose(fCr);

	cout << "Number 6 success" << endl << endl;
//№7
cout << "#7" << endl;
find_min_max(rgb1, h1, w1);
MRGBQUAD** restoredRGB = getRGBfromYreverse(Y, Cb, Cr, h1, w1);
//finish
FILE* frestRGB = fopen("Files\\7\\restored.bmp", "wb");
write_bmp(frestRGB, restoredRGB, &bfh, &bih, h1, w1);
fclose(frestRGB);

cout << "PSNR R = " << get_PSNR_only_R(restoredRGB, r, h1, w1) << endl;
cout << "PSNR G = " << get_PSNR_only_G(restoredRGB, g, h1, w1) << endl;
cout << "PSNR B = " << get_PSNR_only_B(restoredRGB, b, h1, w1) << endl;
cout << std::endl;

//№8 а)
cout << "#8" << endl;
vector<vector<double>> restCb1 = decimation_even_exception(Cb, h1, w1, 2);
vector<vector<double>> restCr1 = decimation_even_exception(Cr, h1, w1, 2);
MRGBQUAD** restoredRGB1 = getRGBfromYreverse(Y, restCb1, restCr1, h1, w1);

//№9
FILE* frestRGB1 = fopen("Files\\9\\restored8a.bmp", "wb");
write_bmp(frestRGB1, restoredRGB1, &bfh, &bih, h1, w1);
fclose(frestRGB1);

//№8 б)

vector<vector<double>> restCb2 = decimation_by_arithmetic_mean(Cb, h1, w1, 2);
```

```
vector<vector<double>> restCr2 = decimation_by_arithmetic_mean(Cr, h1, w1, 2);
MRGBQUAD** restoredRGB2 = getRGBfromYreverse(Y, restCb2, restCr2, h1, w1);

//№9
cout << "#9" << endl;
FILE* frestRGB2 = fopen("Files\\9\\restored8b.bmp", "wb");
write_bmp(frestRGB2, restoredRGB2, &bfh, &bih, h1, w1);
fclose(frestRGB2);

//№10
cout << "#10" << endl;
cout << "Decimation = 2 (a)" << endl;
cout << "YCbCr" << endl;
cout << "PSNR Cb = " << PSNR(Cb, restCb1) << std::endl;
cout << "PSNR Cr = " << PSNR(Cr, restCr1) << std::endl;
cout << "RGB" << std::endl;
cout << "PSNR R = " << get_PSNR_only_R(restoredRGB1, r, h1, w1) << std::endl;
cout << "PSNR G = " << get_PSNR_only_G(restoredRGB1, g, h1, w1) << std::endl;
cout << "PSNR B = " << get_PSNR_only_B(restoredRGB1, b, h1, w1) << std::endl;
cout << endl;

cout << "Decimation = 2 (b)" << endl;
cout << "YCbCr" << endl;
cout << "PSNR Cb = " << PSNR(Cb, restCb2) << std::endl;
cout << "PSNR Cr = " << PSNR(Cr, restCr2) << std::endl;
cout << "RGB" << std::endl;
cout << "PSNR R = " << get_PSNR_only_R(restoredRGB2, r, h1, w1) << std::endl;
cout << "PSNR G = " << get_PSNR_only_G(restoredRGB2, g, h1, w1) << std::endl;
cout << "PSNR B = " << get_PSNR_only_B(restoredRGB2, b, h1, w1) << std::endl;
//11 a)
cout << "\n#11" << endl;
vector<vector<double>> restCb3 = decimation_even_exception(Cb, h1, w1, 4);
vector<vector<double>> restCr3 = decimation_even_exception(Cr, h1, w1, 4);
MRGBQUAD** restoredRGB3 = getRGBfromYreverse(Y, restCb3, restCr3, h1, w1);
FILE* frestRGB3 = fopen("Files\\11\\restored11a.bmp", "wb");
write_bmp(frestRGB3, restoredRGB3, &bfh, &bih, h1, w1);
fclose(frestRGB3);

//11 б)
vector<vector<double>> restCb4 = decimation_by_arithmetic_mean(Cb, h1, w1, 4);
vector<vector<double>> restCr4 = decimation_by_arithmetic_mean(Cr, h1, w1, 4);
MRGBQUAD** restoredRGB4 = getRGBfromYreverse(Y, restCb4, restCr4, h1, w1);
FILE* frestRGB4 = fopen("Files\\11\\restored11b.bmp", "wb");
write_bmp(frestRGB4, restoredRGB4, &bfh, &bih, h1, w1);
fclose(frestRGB4);

//pSNR
cout << "Decimation = 4 (a)" << endl;
cout << "YCbCr" << endl;
cout << "PSNR Cb = " << PSNR(Cb, restCb3) << endl;
cout << "PSNR Cr = " << PSNR(Cr, restCr3) <<endl;
cout << "RGB" << endl;
```

```
cout << "PSNR R = " << get_PSNR_only_R(restoredRGB3, r, h1, w1) << std::endl;
cout << "PSNR G = " << get_PSNR_only_G(restoredRGB3, g, h1, w1) << std::endl;
cout << "PSNR B = " << get_PSNR_only_B(restoredRGB3, b, h1, w1) << std::endl;
cout << endl;

cout << "Decimation = 4 (b)" << endl;
cout << "YCbCr" << endl;
cout << "PSNR Cb = " << PSNR(Cb, restCb4) << endl;
cout << "PSNR Cr = " << PSNR(Cr, restCr4) << endl;
cout << "RGB" << endl;
cout << "PSNR R = " << get_PSNR_only_R(restoredRGB4, r, h1, w1) << std::endl;
cout << "PSNR G = " << get_PSNR_only_G(restoredRGB4, g, h1, w1) << std::endl;
cout << "PSNR B = " << get_PSNR_only_B(restoredRGB4, b, h1, w1) << std::endl;

//№12

cout << "#12" << endl;
file_write("Files\\12\\R.txt", r, h1, w1);
file_write("Files\\12\\G.txt", g, h1, w1);
file_write("Files\\12\\B.txt", b, h1, w1);
file_write("Files\\12\\Y.txt", Y, h1, w1);
file_write("Files\\12\\Cb.txt", Cb, h1, w1);
file_write("Files\\12\\Cr.txt", Cr, h1, w1);*/

//№13
cout << "\n#13" << endl;
std::cout << "H_R = " << entropy(r) << std::endl;
std::cout << "H_G = " << entropy(g) << std::endl;
std::cout << "H_B = " << entropy(b) << std::endl;
std::cout << "H_Y = " << entropy(Y) << std::endl;
std::cout << "H_Cb = " << entropy(Cb) << std::endl;
std::cout << "H_Cr = " << entropy(Cr) << "\n" << std::endl;

//№14
cout << "\n#14" << endl;
//1 - сосед слева
std::vector<std::vector<double>> dpcm_r_1 = DPCM1(r);
std::vector<std::vector<double>> dpcm_g_1 = DPCM1(g);
std::vector<std::vector<double>> dpcm_b_1 = DPCM1(b);
std::vector<std::vector<double>> dpcm_Y_1 = DPCM1(Y);
std::vector<std::vector<double>> dpcm_Cb_1 = DPCM1(Cb);
std::vector<std::vector<double>> dpcm_Cr_1 = DPCM1(Cr);

//2-сосед сверху
std::vector<std::vector<double>> dpcm_r_2 = DPCM2(r);
std::vector<std::vector<double>> dpcm_g_2 = DPCM2(g);
std::vector<std::vector<double>> dpcmb2 = DPCM2(b);
std::vector<std::vector<double>> dpcmY2 = DPCM2(Y);
std::vector<std::vector<double>> dpcmCb2 = DPCM2(Cb);
std::vector<std::vector<double>> dpcmCr2 = DPCM2(Cr);

//3-сосед сверху-слева
```

```
std::vector<std::vector<double>> dpcmr3 = DPCM3(r);
std::vector<std::vector<double>> dpcmg3 = DPCM3(g);
std::vector<std::vector<double>> dpcmb3 = DPCM3(b);
std::vector<std::vector<double>> dpcmY3 = DPCM3(Y);
std::vector<std::vector<double>> dpcmCb3 = DPCM3(Cb);
std::vector<std::vector<double>> dpcmCr3 = DPCM3(Cr);

//4-среднее арифметическое трех соседей
std::vector<std::vector<double>> dpcmr4 = DPCM4(r);
std::vector<std::vector<double>> dpcmg4 = DPCM4(g);
std::vector<std::vector<double>> dpcmb4 = DPCM4(b);
std::vector<std::vector<double>> dpcmY4 = DPCM4(Y);
std::vector<std::vector<double>> dpcmCb4 = DPCM4(Cb);
std::vector<std::vector<double>> dpcmCr4 = DPCM4(Cr);

//№15
cout << "\n#15" << endl;
file_write("Files\\15\\1\\DPCM_R_1.txt", dpcm_r_1, h1, w1);
file_write("Files\\15\\1\\DPCM_G_1.txt", dpcm_g_1, h1, w1);
file_write("Files\\15\\1\\DPCM_B_1.txt", dpcm_b_1, h1, w1);
file_write("Files\\15\\1\\DPCM_Y_1.txt", dpcm_Y_1, h1, w1);
file_write("Files\\15\\1\\DPCM_Cb_1.txt", dpcm_Cb_1, h1, w1);
file_write("Files\\15\\1\\DPCM_Cr_1.txt", dpcm_Cr_1, h1, w1);

file_write("Files\\15\\2\\DPCM_R_2.txt", dpcm_r_2, h1, w1);
file_write("Files\\15\\2\\DPCM_G_2.txt", dpcm_g_2, h1, w1);
file_write("Files\\15\\2\\DPCM_B_2.txt", dpcmb2, h1, w1);
file_write("Files\\15\\2\\DPCM_Y_2.txt", dpcmY2, h1, w1);
file_write("Files\\15\\2\\DPCM_Cb_2.txt", dpcmCb2, h1, w1);
file_write("Files\\15\\2\\DPCM_Cr_2.txt", dpcmCr2, h1, w1);

file_write("Files\\15\\3\\DPCM_R_3.txt", dpcmr3, h1, w1);
file_write("Files\\15\\3\\DPCM_G_3.txt", dpcmg3, h1, w1);
file_write("Files\\15\\3\\DPCM_B_3.txt", dpcmb3, h1, w1);
file_write("Files\\15\\3\\DPCM_Y_3.txt", dpcmY3, h1, w1);
file_write("Files\\15\\3\\DPCM_Cb_3.txt", dpcmCb3, h1, w1);
file_write("Files\\15\\3\\DPCM_Cr_3.txt", dpcmCr3, h1, w1);

file_write("Files\\15\\4\\DPCM_R_4.txt", dpcmr4, h1, w1);
file_write("Files\\15\\4\\DPCM_G_4.txt", dpcmg4, h1, w1);
file_write("Files\\15\\4\\DPCM_B_4.txt", dpcmb4, h1, w1);
file_write("Files\\15\\4\\DPCM_Y_4.txt", dpcmY4, h1, w1);
file_write("Files\\15\\4\\DPCM_Cb_4.txt", dpcmCb4, h1, w1);
file_write("Files\\15\\4\\DPCM_Cr_4.txt", dpcmCr4, h1, w1);

//№16
cout << "\n#16" << endl;
std::cout << "H_DPCM_1_R = " << entropy(dpcm_r_1) << std::endl;
std::cout << "H_DPCM_1_G = " << entropy(dpcm_g_1) << std::endl;
std::cout << "H_DPCM_1_B = " << entropy(dpcm_b_1) << std::endl;
std::cout << "H_DPCM_1_Y = " << entropy(dpcm_Y_1) << std::endl;
std::cout << "H_DPCM_1_Cb = " << entropy(dpcm_Cb_1) << std::endl;
```

```
std::cout << "H_DPCM_1_Cr = " << entropy(dpcm_Cr_1) << "\n" << std::endl;

std::cout << "H_DPCM_2_R = " << entropy(dpcm_r_2) << std::endl;
std::cout << "H_DPCM_2_G = " << entropy(dpcm_g_2) << std::endl;
std::cout << "H_DPCM_2_B = " << entropy(dpcmb2) << std::endl;
std::cout << "H_DPCM_2_Y = " << entropy(dpcmY2) << std::endl;
std::cout << "H_DPCM_2_Cb = " << entropy(dpcmCb2) << std::endl;
std::cout << "H_DPCM_2_Cr = " << entropy(dpcmCr2) << "\n" << std::endl;

std::cout << "H_DPCM_3_R = " << entropy(dpcmr3) << std::endl;
std::cout << "H_DPCM_3_G = " << entropy(dpcmg3) << std::endl;
std::cout << "H_DPCM_3_B = " << entropy(dpcmb3) << std::endl;
std::cout << "H_DPCM_3_Y = " << entropy(dpcmY3) << std::endl;
std::cout << "H_DPCM_3_Cb = " << entropy(dpcmCb3) << std::endl;
std::cout << "H_DPCM_3_Cr = " << entropy(dpcmCr3) << "\n" << std::endl;

std::cout << "H_DPCM_4_R = " << entropy(dpcmr4) << std::endl;
std::cout << "H_DPCM_4_G = " << entropy(dpcmg4) << std::endl;
std::cout << "H_DPCM_4_B = " << entropy(dpcmb4) << std::endl;
std::cout << "H_DPCM_4_Y = " << entropy(dpcmY4) << std::endl;
std::cout << "H_DPCM_4_Cb = " << entropy(dpcmCb4) << std::endl;
std::cout << "H_DPCM_4_Cr = " << entropy(dpcmCr4) << "\n" << std::endl;

//№17
cout << "\n#17" << endl;
MRGBQUAD** reflection = horizonral_reflection(rgb1, h1, w1);
FILE* file_reflection = fopen("Files\\17\\reflection.bmp", "wb");
write_bmp(file_reflection, reflection, &bfh, &bih, h1, w1);
fclose(file_reflection);
}
```