МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ТВ

ОТЧЁТ

по лабораторной работе № 5

по дисциплине «Цифровая обработка изображений»

Тема: РАЗРАБОТКА ПРОГРАММЫ ДЛЯ СЖАТИЯ ЦИФРОВЫХ ИЗОБРАЖЕНИЙ

Студенты гр. 9105 _____ Шаривзянов Д. Р.

_____ Басманов А. А.

Преподаватель _____ Поздеев А. А.

Санкт-Петербург

2024

РАЗРАБОТКА ПРОГРАММЫ ДЛЯ СЖАТИЯ ЦИФРОВЫХ ИЗОБРАЖЕНИЙ

Код программы:

```
#include <iostream>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <vector>

using namespace std;
using namespace cv;

struct Image {
   Mat bgr;
   Mat gray;

   int compression;

   Mat dct_orig;
   Mat dct_decoded;
   Mat gray_decoded;
};

Mat getHist(const Mat &src) {
   int hist_h = 400, hist_w = 256*3;

         Mat hist = Mat::zeros(1, 256, CV_64FC1);

         for (int i = 0; i < src.cols; i++)
               for (int j = 0; j < src.rows; j++) {
                     int r = src.at<unsigned char>(j, i);
                     hist.at<double>(0, r) = hist.at<double>(0, r) + 1.0;
               }
         double m = 0, M = 0;
         minMaxLoc(hist, &m, &M);
         hist = hist / M;
         Mat hist_img = Mat::zeros(100, 256, CV_8U);
         for (int i = 0; i < 256; i++)
               for (int j = 0; j < 100; j++) {
                     if (hist.at<double>(0, i) * 100 > j) {
                           hist_img.at<unsigned char>(99 - j, i) = 255;
                     }
               }
         bitwise_not(hist_img, hist_img);
   resize(hist_img, hist_img, Size(hist_w, hist_h), 0 , 0, INTER_NEAREST);
         return hist_img;
}

vector<double> probality(const Mat &src) {
   vector<double> prob(256);
   for (int i = 0; i < src.cols; i++)
      for (int j = 0; j < src.rows; j++) {
         int r = src.at<uchar>(j, i);
         prob[r] += 1.0;
          }

   for (int i = 0; i < prob.size(); i++) prob[i] /= (src.rows*src.cols);
   return prob;
}
```

```
double entropy(vector<double> &prob) {
   double H = 0.0;
   for (int i = 0; i < prob.size(); i++)
      if (prob[i] != 0) H -= prob[i] * log2(prob[i]);
   return H;
}

double redundancy(double H) {
   return 1 - (H / log2(256));
}

void gammaDivision(Mat &dst, Mat &gamma, int block_size) {
   for (int row = 0; row < block_size; row++)
      for (int col = 0; col < block_size; col++)
         dst.at<double>(row, col) /= gamma.at<double>(row, col);
}

void gammaMultiply(Mat &dst, Mat &gamma, int block_size) {
   for (int row = 0; row < block_size; row++)
      for (int col = 0; col < block_size; col++)
         dst.at<double>(row, col) *= gamma.at<double>(row, col);
}

void makeGamma(Mat &dst, int quality, int block_size) {
   dst = Mat::zeros(block_size, block_size, CV_64FC1);
   for (int row = 0; row < block_size; row++)
      for (int col = 0; col < block_size; col++)
         dst.at<double>(row, col) = block_size + (row + col) * quality;
}

void createBasisMat(Mat &basisMat, int block_size) {
   basisMat = Mat::zeros(block_size, block_size, CV_64FC1);
   for (int row = 0; row < block_size; row++)
      for (int col = 0; col < block_size; col++) {
         if (row == 0) basisMat.at<double>(row, col) = 1 / sqrt(block_size);
         else if (row > 0) basisMat.at<double>(row, col) = sqrt(2. / block_size) * cos(((CV_PI * row) / block_size) *
(col + 0.5));
      }
}

void DCT_direct(Mat &src, Mat &dst, int quality) {
   int block_size = 8;
   int delta_h8 = src.rows % block_size;
   int delta_w8 = src.cols % block_size;
   dst = Mat::zeros(src.rows - delta_h8, src.cols - delta_w8, CV_8UC1);

   Mat img_flt;
   src.convertTo(img_flt, CV_64FC1);

   Mat gamma;
   makeGamma(gamma, quality, block_size);

   Mat basisMat;
   createBasisMat(basisMat, block_size);

   //перебираем изображение по блокам 8x8
   for (int block_row = 0; block_row < dst.rows; block_row += block_size)
      for (int block_col = 0; block_col < dst.cols; block_col += block_size) {

         //выделяем область 8x8 исходного изображения
         Mat ROI8U = img_flt(Rect(block_col, block_row, block_size, block_size));
```

```
            // Преобразуем в double для ДКП
            Mat ROI64F;
            ROI8U.convertTo(ROI64F, CV_64FC1);

            //применяем DCT
            Mat DCT64F = basisMat * ROI64F * basisMat.t();

            //применяем гамму
            gammaDivision(DCT64F, gamma, block_size);

            //собираем изображение по блокам 8x8
            DCT64F.copyTo(dst(Rect(block_col, block_row, block_size, block_size)));
        }
    // Конвертируем в 8 бит
    dst.convertTo(dst, CV_8UC1);
}

void DCT_inverse(Mat &src, Mat &dst, int quality) {
    int block_size = 8;
    dst = Mat::zeros(src.size(), CV_64FC1);

    Mat gamma;
    makeGamma(gamma, quality, block_size);

    Mat basisMat;
    createBasisMat(basisMat, block_size);

    // Перебираем изображение по блокам 8x8
    for (int block_row = 0; block_row < src.rows; block_row += block_size)
        for (int block_col = 0; block_col < src.cols; block_col += block_size) {

            // Выделяем область 8x8 из DCT-изображения
            Mat DCT8U = src(Rect(block_col, block_row, block_size, block_size));

            // Преобразуем в double для обратного ДКП
            Mat DCT64F;
            DCT8U.convertTo(DCT64F, CV_64FC1);

            // Применяем обратную гамму
            gammaMultiply(DCT64F, gamma, block_size);

            // Выполняем обратное ДКП
            Mat ROI64F = basisMat.t() * DCT64F * basisMat;

            // Собираем изображение по блокам 8x8
            ROI64F.copyTo(dst(Rect(block_col, block_row, block_size, block_size)));
        }
    // Конвертируем обратно в 8 бит
    dst.convertTo(dst, CV_8UC1);
}

vector<uchar> block2vec(const Mat& block) {
    const uchar n = 8; // Размер блока 8x8
    vector<uchar> output(n * n, 0);
    int i = 0, j = 0;
    bool goingUp = true; // Флаг направления движения

    for (int k = 0; k < n * n; k++) {
        output[k] = block.at<uchar>(i, j); // Использование типа uchar
        if (goingUp) {
            if (j == n - 1) {
                i++; // Двигаемся вниз, если достигли правой границы
```

```
            goingUp = false;
        } else if (i == 0) {
            j++; // Двигаемся вправо, если достигли верхней границы
            goingUp = false;
        } else {
            i--;
            j++;
        }
    } else {
        if (i == n - 1) {
            j++; // Двигаемся вправо, если достигли нижней границы
            goingUp = true;
        } else if (j == 0) {
            i++; // Двигаемся вниз, если достигли левой границы
            goingUp = true;
        } else {
            i++;
            j--;
        }
    }
  }

  return output;
}

vector<uchar> mat2vec(const Mat& src) {
  vector<uchar> output;
  for (int i = 0; i < src.rows; i += 8) {
    for (int j = 0; j < src.cols; j += 8) {
      vector<uchar> block = block2vec(src(Rect(j, i, 8, 8)));
      output.insert(output.end(), block.begin(), block.end());
    }
  }
  return output;
}

vector<pair<uchar, uchar>> vec2RLE(vector<uchar>& data, int blockSize = 64) {
  vector<pair<uchar, uchar>> encodedData;
  int count = 0;
  int prevValue = (data.empty() ? -1 : data[0]);

  for (int i = 0; i < data.size(); ++i) {
    if (data[i] == prevValue) {
      count++;
    } else {
      if (count > 0) {
        encodedData.emplace_back(count, prevValue);
      }
      prevValue = data[i];
      count = 1;
    }

    // Вставляем код конца блока после обработки каждого блока 8x8
    if ((i + 1) % blockSize == 0) {
      if (count > 0) {
        encodedData.emplace_back(count, prevValue);
        count = 0;
      }
      // Код конца блока
      encodedData.emplace_back(255, 255);
      prevValue = (i + 1 < data.size() ? data[i + 1] : -1);
    }
```

```
    }

    // Добавляем оставшиеся данные, если они есть
    if (count > 0 && (data.size() % blockSize) != 0) {
        encodedData.emplace_back(count, prevValue);
    }

    return encodedData;
}

vector<uchar> RLE2vec(vector<pair<uchar, uchar>>& rle) {
    vector<uchar> decoded;
    for (const auto& pair : rle) {
        if (pair.first == 255 && pair.second == 255) {
            // Окончание блока, но продолжаем обработку, если есть еще данные
            continue;
        }
        for (int i = 0; i < pair.first; i++) {
            decoded.push_back(pair.second);
        }
    }
    return decoded;
}

// Функция для преобразования зигзаг-последовательности в блок 8x8
Mat vec2block(vector<uchar>& zigzag, int startIdx) {
    Mat block(8, 8, CV_8UC1);  // Используем double для хранения значений
    vector<uchar> indexMap = {
        0,  1,  5,  6, 14, 15, 27, 28,
        2,  4,  7, 13, 16, 26, 29, 42,
        3,  8, 12, 17, 25, 30, 41, 43,
        9, 11, 18, 24, 31, 40, 44, 53,
        10, 19, 23, 32, 39, 45, 52, 54,
        20, 22, 33, 38, 46, 51, 55, 60,
        21, 34, 37, 47, 50, 56, 59, 61,
        35, 36, 48, 49, 57, 58, 62, 63
    };

    for (int i = 0; i < 64; ++i) {
        int x = i / 8;
        int y = i % 8;
        block.at<uchar>(x, y) = zigzag[startIdx + indexMap[i]];
    }

    return block;
}

// Функция для создания изображения из всех блоков
Mat vec2mat(vector<uchar>& zigzag, int width, int height) {
    CV_Assert(width % 8 == 0 && height % 8 == 0);
    Mat image(height, width, CV_8UC1);

    int blocksPerRow = width / 8;
    int blocksPerColumn = height / 8;
    int index = 0;

    for (int i = 0; i < blocksPerColumn; ++i) {
        for (int j = 0; j < blocksPerRow; ++j) {
            Mat block = vec2block(zigzag, index);
            block.copyTo(image(Rect(j * 8, i * 8, 8, 8)));
            index += 64;
        }
```

```
    }

    return image;
}

void lab5(const Mat &img_bgr) {
    Image img;
    img.bgr = img_bgr;
    // resize(img.bgr, img.bgr, Size(300, 300), 0, 0, INTER_CUBIC);
    imshow("image bgr", img.bgr);

    cvtColor(img.bgr, img.gray, COLOR_BGR2GRAY);
    imshow("image gray", img.gray);
    imwrite("../../Images/Lab 5/image gray.jpg", img.gray);

    vector <double> prob_orig = probality(img.gray);
    double H_orig = entropy(prob_orig);
    cout << "H ref = " << log2(256) << '\t';
    cout << "H orig = " << H_orig << '\t';

    double R = redundancy(H_orig);
    cout << "R orig = " << R << endl;

    //-------------------------------------вычисление DCT------------------------------
    img.compression = 5;
    DCT_direct(img.gray, img.dct_orig, img.compression);
    imshow("image dct", img.dct_orig);
    imwrite("../../Images/Lab 5/image dct.jpg", img.dct_orig);

    Mat hist = getHist(img.dct_orig);
    imshow("Histogram", hist);
    imwrite("../../Images/Lab 5/histogram.jpg", hist);

    vector <double> prob_dct = probality(img.dct_orig);
    double H_dct = entropy(prob_dct);
    cout << "H dct = " << H_dct << '\t';

    double R_dct = redundancy(H_dct);
    cout << "R dct = " << R_dct << endl;
    //-------------------------------------------кодирование-----------------------------------------
    cout << "Total bytes " << img.dct_orig.total() << endl;

    vector <uchar> vec_orig = mat2vec(img.dct_orig);
    cout << "ZigZag origin bytes " << vec_orig.size() << endl;

    vector<pair<uchar, uchar>> RLE = vec2RLE(vec_orig);
    cout << "Encoded bytes " << RLE.size() * 2 << endl;
    cout << "Compression ratio " << double(vec_orig.size()) / (RLE.size() * 2) << endl;
    //-------------------------------------------декодирование---------------------------------------
    vector<uchar> vec_decoded = RLE2vec(RLE);
    cout << "ZigZag decoded bytes " << vec_decoded.size() << endl;

    img.dct_decoded = vec2mat(vec_decoded, img.dct_orig.cols, img.dct_orig.rows);
    imshow("image dct decoded", img.dct_decoded);
    imwrite("../../Images/Lab 5/image dct decoded.jpg", img.dct_decoded);

    DCT_inverse(img.dct_decoded, img.gray_decoded, img.compression);
    imshow("image gray decoded", img.gray_decoded);
    imwrite("../../Images/Lab 5/image gray decoded.jpg", img.gray_decoded);

    waitKey();
}
```

Исходное изображение представлено на рис. 1.

Рис. 1. Портрет.

Ход работы.

1. Оценим возможности по энтропийному кодированию исходного изображения, квантованного на 256 уровней. Используем данные гистограммы распределения уровней яркости и просчитаем вероятности появления этих уровней; энтропию источника сообщения, а также избыточность сообщений.

Энтропия: Horig = 7.54554

Избыточность: Rorig = 0.0568071

2. Построим гистограмму (рис. 2) появления коэффициентов ДКП по всем блокам; просчитаем энтропию источника сообщения, а также избыточность сообщений.

Энтропия: Hdct = 0.491461

Избыточность: Rdct = 0.938567

Рис. 2. Гистограмма появления коэффициентов ДКП по всем блокам.

На рис. 3 представлено изображение после ДКП в каждом блоке.

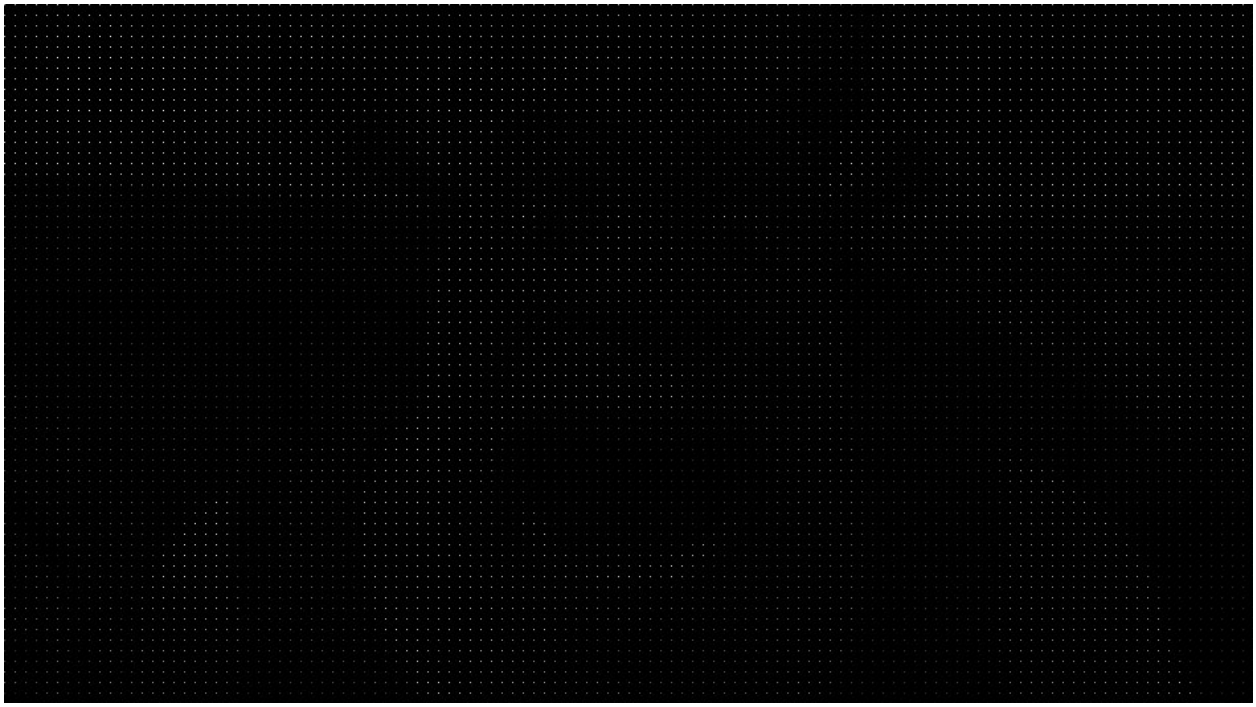

Рис. 3. Блоки ДКП.

3. Используя функцию makeGamma, сгенерируем кодовую таблицу и поделим на неё коэффициенты ДКП в блоках, округлив результаты. Выполним зигзаг-сканирование коэффициентов в блоках и RLE-кодирование. Подсчитаем число бит закодированного «Портрета» и оценим полученный выигрыш:

исходное изображение содержит 498432 байт;

результат зигзаг-сканирования содержит 498432 байт;

результат RLE-кодирования содержит 93944 байт;

выигрыш составляет 5.30563 раза.

6. Восстановим изображение и сравните с оригиналом (рис. 4 и рис. 5).

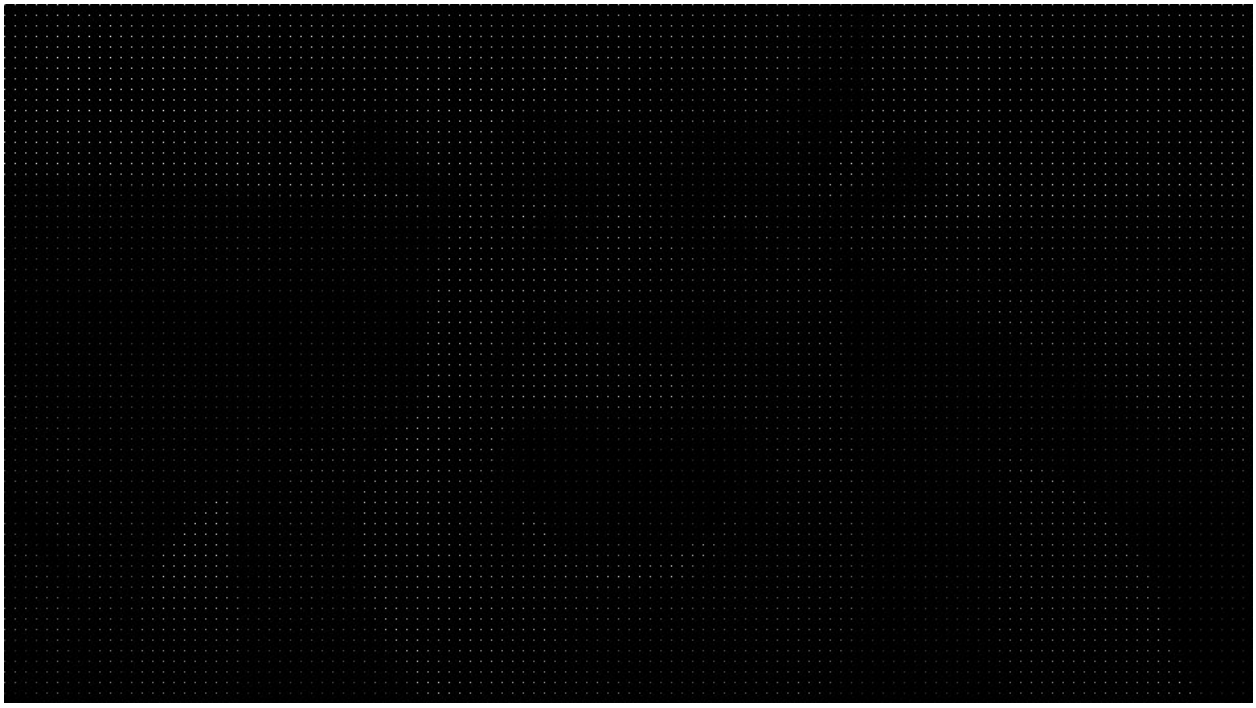

Рис. 4. Блоки ДКП после восстановления.

Рис. 5. Изображение после восстановления.

Выводы.

В ходе данной лабораторной работы была разработана программа, которая способна сжимать изображение от 3 до 10 раз в зависимости от входного параметра quality. Сжатие осуществляется во многом за счёт удаления высокочастотных компонент из блоков ДКП, последствием такого метода является наличие блочных структур на восстановленном изображении (см. рис. 5).