Министерство науки и высшего образования РФ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет автоматики и вычислительной техники

Кафедра радиоэлектронных средств

Отчёт по лабораторной работе №2

**Обработка изображений**

дисциплина «Проектирование устройств и систем на цифровых сигнальных процессорах»

|  |  |  |
| --- | --- | --- |
| Выполнил студент гр.ИНБс-5301: |  | /И.И. Золотаева/ |
| Проверил: старший преподаватель кафедры систем автоматизации управления |  | /М.А. Земцов/ |

Киров 2023

**Цель:** изучение дискретного косинусного преобразования изображения; получение навыков обработки изображений на DSK.

**Ход работы**

**1 Формирование файла-пробника**

В первую очередь необходимо преобразовать полутоновое изображение в удобный для обработки формат. Реализовано это через скрипт Python:

import numpy

from PIL import Image

def get\_image(image\_path):

"""Get a numpy array of an image so that one can access values[x][y]."""

image = Image.open(image\_path, "r")

width, height = image.size

pixel\_values = list(image.getdata())

if image.mode == "RGB":

channels = 3

elif image.mode == "L":

channels = 1

else:

print("Unknown mode: %s" % image.mode)

return None

pixel\_values = numpy.array(pixel\_values).reshape((width, height, channels))

out = []

for i in range(len(pixel\_values)):

for j in range(len(pixel\_values[i])):

out.append(pixel\_values[i][j][0])

return out

bits = get\_image("D:\\5 курс\\Земцов\\image1.bmp")

f = open("D:\\5 курс\\Земцов\\bits.txt", 'w')

for i in range(len(bits)):

f.write(str(bits[i])+"\n")

f.close()



Рисунок 1 – Исходное изображение



Рисунок 2 – Данные для обработки

**2 Обработка данных**

Далее, необходимо преобразовать входные данные по алгоритму дискретного косинусного преобразования в частотную форму. Для загрузки файла из которого данные будут читаться в память, а также для записи в файл, выставим Probe Point до и после реализации алгоритма DCT:

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <stdint.h>

#define PI 3.141592653589793

#define LENGTH 64

void dataInput() {

return;

}

void dataOutput() {

return;

}

void prepareDataToInput(int\* in, float\* out, int LENGTHBuffer) {

int i;

for(i = 0; i < LENGTHBuffer; i++) {

out[i] = 0 + in[i];

}

}

void prepareDataToOutput(float\* in, int\* out, int LENGTHBuffer) {

int i;

for(i = 0; i < LENGTHBuffer; i++) {

out[i] = (int)in[i];

}

}

void transpon(float \*arr, float\* transpon\_matrix)

{

int m = LENGTH/8;

int n = m;

int i = 0;

int j = 0;

for (i = 0; i < m; ++i )

{

for (j = 0; j < n; ++j )

{

// Index in the original matrix.

int index1 = i\*n+j;

// Index in the transpose matrix.

int index2 = j\*m+i;

transpon\_matrix[index2] = arr[index1];

}

}

}

void multiply(float \*arr1, float \*arr2, float \*after\_multyply) {

int i, j, k;

int tempLENGTH = LENGTH/8;

for (i = 0; i < tempLENGTH; i++) {

for (j = 0; j < tempLENGTH; j++) {

after\_multyply[i \* tempLENGTH + j] = 0;

for (k = 0; k < tempLENGTH; k++) {

after\_multyply[i \* tempLENGTH + j] += arr1[i \* tempLENGTH + k] \* arr2[k \* tempLENGTH + j];

}

}

}

}

void dct(float \*arr){

int row;

int col;

int tempLENGTH = LENGTH/8;

for(row = 0; row < tempLENGTH; row++) {

for(col = 0; col < tempLENGTH; col++) {

if(row == 0) {

arr[row \* tempLENGTH + col] = 1 / sqrt(tempLENGTH);

arr[row \* tempLENGTH + col] \*= cos((2 \* col + 1) \* row \* PI / (2 \* tempLENGTH));

} else {

arr[row \* tempLENGTH + col] = sqrt(2.0 / tempLENGTH);

arr[row \* tempLENGTH + col] \*= cos((2 \* col + 1) \* row \* PI / (2 \* tempLENGTH));

}

}

}

}

void shakal(float\* arr, int shift){

int i = shift;

for (i = shift; i<LENGTH; i++){

arr[i]=0;

}

}

void IDCT(float \*transpon\_matrix, float \*P\_DCT, float \*dct\_matrix, float \* out){

float temp[64];

multiply(transpon\_matrix,P\_DCT, temp);// DCT^T \* P\_DCT

multiply(temp, dct\_matrix,out);

}

void DCT(float\* bits, float\* dct\_matrix,float\* transpon\_matrix, float\* DCTxP, float\* P\_DCT, int shift){

dct(dct\_matrix);//DCT

shakal(dct\_matrix, shift);//DCT shakal

multiply(dct\_matrix, bits,DCTxP);//DCT \* P

transpon(dct\_matrix,transpon\_matrix);//DCT ^ T

multiply(DCTxP,transpon\_matrix, P\_DCT);// DCT ^ T \* DCT \* P

}

int main()

{

int shift = 32; // ?????? shift ???????? ????????, ????????? ???? ? ??????? dct

int inputArr[LENGTH];

int outputArr[LENGTH];

float out[LENGTH];

float bits1d[LENGTH];

float dct\_matrix[LENGTH];//DCT

float transpon\_matrix[LENGTH];

float DCTxP[LENGTH];//DCT \* P

float P\_DCT[LENGTH];//PDCT

float dct\_tranps[LENGTH];

//int LENGTH\_all = 320\*320;

//float bits\_all[320\*320];//P 102 400 elements

int block=0;

for (block=0; block<40\*40; block++){

dataInput();

prepareDataToInput(inputArr, bits1d,64);

///DCT begin

DCT(bits1d, dct\_matrix,transpon\_matrix, DCTxP, P\_DCT,shift);

///IDCT begin

IDCT(transpon\_matrix, P\_DCT, dct\_matrix, out);

prepareDataToOutput(out, outputArr,64);

dataOutput();

}

return 0;

}

**2 Результаты работы**

В результате получились следующие выходные данные (рисунок 3).

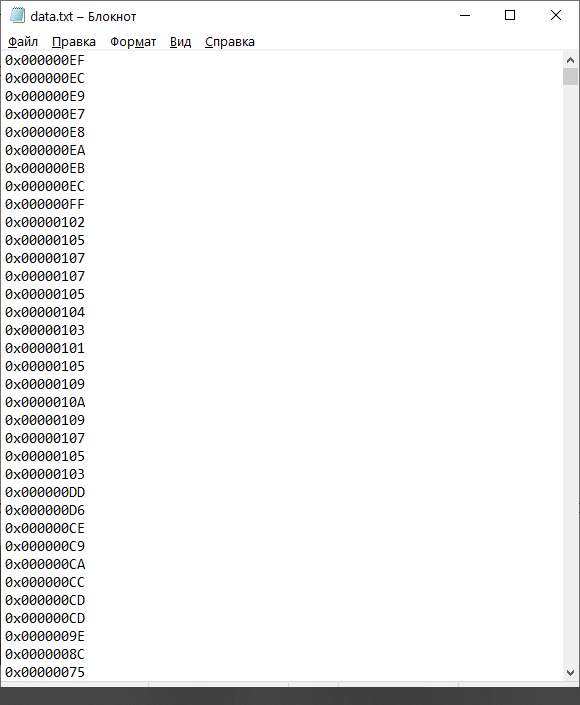


Рисунок 3 – Пример выходных данных

Потом, используя скрипт Python, получаем следующие восстановленные изображения.

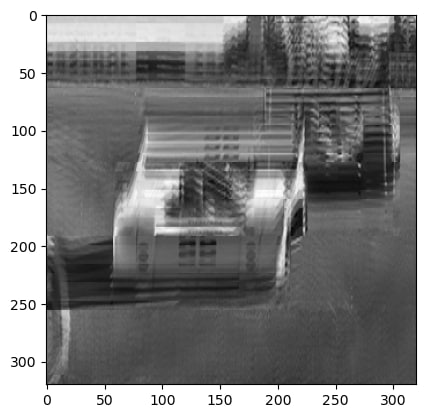


Рисунок 4 – Сжатие до 32 значащих частот из 64

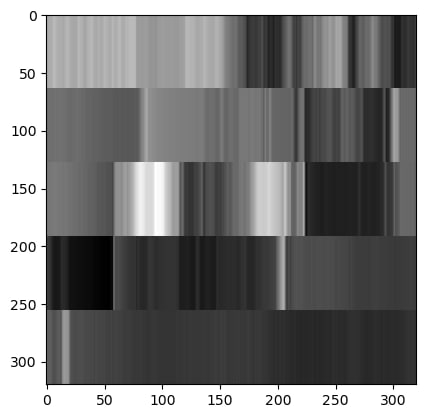


Рисунок 5 – Сжатие до 10 значащих элементов из 64

**Выводы:**

Изображение разрешением 320x320 разбивается на блоки 8x8 пикселей и обрабатывается DCT алгоритмом. Результатом DCT алгоритма является матрица 8x8 со значениями частот, которые убывают слева направо, сверху вниз. Процесс сжатия заключается в занулении менее значимых частот в матрице 8x8 DCT. После зануления методом IDCT идет восстановление изображения из частотной области. Чем больше мы занулим значимых частот, тем сильнее исказиться изображение относительно исходного.

Преимущества дискретного косинусного преобразования изображения:

* Компактность представления: после преобразования изображения преобразование занимает меньше места, чем исходное изображение.
* Сжатие: ДКП можно использовать для сжатия изображения, что позволяет передавать изображения более эффективно и экономично по объему.
* Высокая эффективность: ДКП дает приемлемые результаты при отсутствии ярко выраженной локализации контента.
* Улучшение качества изображения: ДКП может улучшить качество изображения, уменьшив неравномерность его распределения, что может привести к более четкому и улучшенному виду изображения.
* Устойчивость: ДКП устойчив к шуму и другим искажениям входного изображения, что позволяет ему сохранять качество изображения даже при наличии шумов.

Недостатки дискретного косинусного преобразования изображения:

* Низкая сложность вычислений: ДКП требует меньше вычислительных ресурсов, чем некоторые альтернативные методы, что позволяет использовать его для обработки больших объемов изображений в режиме реального времени.
* Неэффективность при обработке больших изображений: DCT может потребовать слишком много времени для обработки больших изображений, особенно если требуется вычислять большое количество коэффициентов.
* Операции с плавающей запятой: в процессе вычисления DCT используются операции с плавающей запятой, которые могут занимать необходимое время и ресурсы.
* Неравномерность пространственной чувствительности: DCT не всегда может правильно отображать чувствительность изображения, так как он не способен регулировать различные области изображения в зависимости от их содержания информации.
* Низкая устойчивость к шуму: DCT часто подвержен шумам при обработке изображения, что может привести к ухудшению качества изображения.
* Необходимость обратного преобразования: для восстановления исходного изображения из коэффициентов DCT необходимо выполнить обратное DCT преобразование, что занимает некоторое время и может замедлить процесс обработки изображения.