###### МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

**Кафедра параллельных вычислений**

ОТЧЕТ

О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

«ВЫСОКОУРОВНЕВАЯ РАБОТА С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ»

студента 2 курса, 18209 группы

**Большим Максима Антоновича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Власенко Андрей Юрьевич

Новосибирск 2019

1. **Цель**

1. Ознакомиться с программированием периферийных устройств на примере ввода данных с Web-камеры с использованием библиотеки OpenCV;

2. Применить полученные знания в реализации программы вывода в окно искаженного изображения с камеры.

**Задание**

Изучить принципы работы OpenCV, ознакомиться с: работой с растровыми изображениями и их форматами, работой с видеоданными и их форматами, интерфейсом для работы с камерами, реализацией упрощенного оконного интерфейса, операций над векторами и матрицами. Реализовать с помощью полученных знаний об этой библиотеке программу, что будет принимать видеосигнал с камеры и искажать полученное изображение путем сдвига цветовых каналов. Замерить время, затрачиваемое на получение, обработку и вывод кадров на окно программы. Провести анализ полученных данных.

**Листинг программы**

#include "opencv2/highgui.hpp"

#include "opencv2/videoio.hpp"

#include <opencv2/opencv.hpp>

#include <ctime>

#include <iostream>

using namespace std;

using namespace cv;

int main(int argc, const char \*\*argv)

{

VideoCapture capture;

Mat frame, blurMask;

capture.open(0);

long frameCount = 0;

long double totalTime = 0, inputTime = 0, procTime = 0, outputTime = 0;

if (capture.isOpened())

{

cout << "Video capturing has been started ..." << endl;

while (true)

{

clock\_t c\_start = clock();

capture >> frame;

clock\_t c\_end = clock();

double timeElapsed0 = 1000.0 \* (c\_end - c\_start) / CLOCKS\_PER\_SEC;

totalTime += timeElapsed0;

inputTime += timeElapsed0;

c\_start = clock();

if (frame.empty())

break;

Mat output(frame.rows, frame.cols, CV\_8UC3, cv::Scalar(0));

Mat blueC(frame.rows, frame.cols, CV\_8UC3);

Mat redC(frame.rows, frame.cols, CV\_8UC3);

Mat greenC(frame.rows, frame.cols, CV\_8UC3);

vector<Mat> rgbChannels(3);

split(frame, rgbChannels);

vector<Mat> channel{rgbChannels[0], Mat::zeros(frame.rows, frame.cols, CV\_8UC1),

Mat::zeros(frame.rows, frame.cols, CV\_8UC1)};

merge(channel, blueC);

channel.clear();

channel = {Mat::zeros(frame.rows, frame.cols, CV\_8UC1), Mat::zeros(frame.rows, frame.cols, CV\_8UC1),

rgbChannels[2]};

merge(channel, redC);

channel.clear();

channel = {Mat::zeros(frame.rows, frame.cols, CV\_8UC1), rgbChannels[1],

Mat::zeros(frame.rows, frame.cols, CV\_8UC1)};

merge(channel, greenC);

cv::Mat redMoved(frame.size(), frame.type(), cv::Scalar::all(0));

redC(cv::Rect(10, 0, frame.cols - 10, frame.rows)).copyTo(

redMoved(cv::Rect(0, 0, frame.cols - 10, frame.rows)));

cv::Mat blueMoved(frame.size(), frame.type(), cv::Scalar::all(0));

blueC(cv::Rect(0, 0, frame.cols - 10, frame.rows)).copyTo(

blueMoved(cv::Rect(10, 0, frame.cols - 10, frame.rows)));

addWeighted(output, 1, redMoved, 1, 1.5, output);

addWeighted(output, 1, greenC, 1, 1.5, output);

addWeighted(output, 1, blueMoved, 1, 1.5, output);

GaussianBlur(output, blurMask, cv::Size(0, 0), 15);

addWeighted(output, 1.5, blurMask, -0.5, 0, output);

c\_end = clock();

double timeElapsed1 = 1000.0 \* (c\_end - c\_start) / CLOCKS\_PER\_SEC;

totalTime += timeElapsed1;

procTime += timeElapsed1;

c\_start = clock();

imshow("result", output);

c\_end = clock();

double timeElapsed2 = 1000.0 \* (c\_end - c\_start) / CLOCKS\_PER\_SEC;

outputTime += timeElapsed2;

totalTime += timeElapsed2;

frameCount++;

char c = (char) waitKey(1);

if (c == 27 || c == 'q' || c == 'Q')

break;

}

}

double procent = totalTime / 100.0;

cout << "CPU time for reading frames: " << inputTime / procent << "%" << endl;

cout << "CPU time for processing frames: " << procTime / procent << "%" << endl;

cout << "CPU time for output frames: " << outputTime / procent << "%" << endl;

cout << "Total framerate: " << ((frameCount - 1) / (totalTime / 1000.0)) << endl;

return 0;

}

**Оценка времени работы**

Ниже приведен один из выводов программы:

Video capturing has been started ...

CPU time for reading frames: 1.52854%

CPU time for processing frames: 96.6518%

CPU time for output frames: 1.81968%

Total framerate: 3.58503

Эти данные отображают процентное соотноение времени, затрачиваемого на ввод, обработку и вывод изображения на экран. На основе данных можно сказать, что в секунду программа прогоняет около 3-4 кадров в секунду. Такая маленькая скорость обусловлена следующими факторами:

Устаревшая видеокамера;

Огромное количество работы с матрицами в программе и ручная постобработка;

Работа с матрицами и постобработкой происходила в процессоре, а не на графическом ускорителе.

**Выводы**

В ходе работы с OpenCV были освоены:

работа с камерой;

работа с изображениями, полученными с камеры;

работа с матрицами изображений, альфа-каналами и цветовыми каналами, Гауссовым размытием изображений и сдвигом изображений;

вывод обработанного изображения на экран и работа с окнами в OpenCV.