МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №5
по курсу «ЭВМ и периферийные устройства»

НИЗКОУРОВНЕВАЯ РАБОТА С ПЕРИФЕРИЙНЫМИ УСТРОЙСТВАМИ

Выполнил: студент 2-го курса гр. 17208
Гафиятуллин А.Р.

Новосибирск, 2018

1. ЦЕЛИ РАБОТЫ:

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

2. ХОД РАБОТЫ:

1. Для достижения поставленных целей была написана программа с использованием библиотеки OpenCV, осуществляющая ввод изображения с Web-камеры и добавляющая на него мерцающий, меняющий свой цвет с различной периодичностью, зашумленный масонский треугольник с глазом.
2. Компиляция и тестирование программы проходили на Linux-машине с Elementary OS 64 bit: Linux kernel 4.15.0-36-generic, Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, встроенная Web-камера с разрешением 640x480, выдающая ≈ 15 fps (без обработки изображения в программе для тестирования веб-камеры из поставки дистрибутива). На момент тестирования программы в ОС было запущено около 180 процессов.
3. Исходный код программы:

```
#include <opencv2/highgui/highgui.hpp>  //для функций OpenCV
#include <time.h>                       //для time()
#include <stdlib.h>                     //для rand()
#include <stdio.h>                      //для printf()
#include <sys/times.h>                  //для times()
#include <unistd.h>                     //для sysconf()
#define NOISE_LEVEL 40                  //уровень шума

//выражение области для закраски через
уравнения прямых с заданными угловыми
коэффициентами
int construct_central_triangle(IplImage* image, int x, int y, int x_coeff,
int y_coeff){
    return (y_coeff * (image->height - y) <= x_coeff * x &&
```

```
            x_coeff * (x - image->width / 2) <= y_coeff * y);
}

//выражение области для закраски через
уравнения окружностей с заданными радиусами в
квадрате
int construct_central_eye(IplImage* image, int x, int y,
int squared_top_eyelid_rad, int squared_bot_eyelid_rad){
    return (x - image->width / 2) * (x - image->width / 2) +
            y * y <= squared_top_eyelid_rad && (x - image->width / 2) *
            (x - image->width / 2) + (y - image->height)
            * (y - image->height) <= squared_bot_eyelid_rad;
}

//выражение области для закраски через
уравнение окружности /с заданным радиусом
int construct_central_pupil(IplImage* image, int x, int y, int radius){
    return (x - image->width / 2) * (x - image->width / 2)
            + (y - image->height / 2) * (y - image->height / 2) >= radius;
}

int masonic_triangle(IplImage* image, int x, int y){
    //простая математика для вычисления
необходимых радиусов
    int bot_eyelid_rad_in_square = ((image->height * image->height)
        + (image->width / 2 * image->width / 2)) / 4;
    int top_eyelid_rad_in_square = (image->width / 4)
        * (image->width / 4) + (image->height / 2) * (image->height / 2);
    return  construct_central_triangle(image, x, y, image->height,
            image->width / 2) && !(construct_central_eye(image, x, y,
            top_eyelid_rad_in_square, bot_eyelid_rad_in_square) &&
            construct_central_pupil(image, x, y, (image->width / 16)
            * (image->width / 16)));
}
```

```
int main(){
    struct tms start, finish, start_processing_time,
        finish_processing_time, start_input_time, finish_input_time,
        finish_output_time;
    long long int clocks_per_sec = sysconf(_SC_CLK_TCK);
    //подсчет полного затраченного
процессорного времени
    times(&start);
    //подсчет процессорного времени,
затраченного на преобразование изображения,
    //ввод и вывод видеоданных
    double total_processing_time = 0, total_input_time = 0, total_output_time
= 0;
    int i = 0, frame_delay = 0;
    long long int frames_amount = 0;
    srand(time(NULL));
    CvCapture *capture = cvCreateCameraCapture(0);
    if (!capture) return 0;
    //подсчет времени работы цикла для
измерения fps
    long long int execute_time_start = time(NULL);
    while(1){
        frames_amount++;
        CvRNG rng = cvRNG(time(NULL));
        //генерация случайного значения для
задержки
        if(!i) frame_delay = rand() % 10;
        times(&start_input_time);
        IplImage *frame = cvQueryFrame(capture);
        times(&finish_input_time);
        total_input_time += finish_input_time.tms_utime -
start_input_time.tms_utime;
        if(!frame) break;
```

```
        times(&start_processing_time);
        IplImage *image = cvCloneImage(frame);
        if(i != frame_delay)
        for (int y = 0; y < image->height; y++){
                uchar *ptr = (uchar*)(image->imageData + y *
image->widthStep);
                for(int x = 0; x < image->width; x++){
                    if(masonic_triangle(image, x, y)
                        && cvRandInt(&rng) % 100 >= NOISE_LEVEL){
                            //создание шума
                            ptr[3 * x] = cvRandInt(&rng) % 255;
                            ptr[3 * x + 1] = cvRandInt(&rng) % 255;
                            ptr[3 * x + 2] = cvRandInt(&rng) % 255;
                    }
                }
                cvCircle(image, cvPoint(image->width / 2, image->height / 2),
                    image->width / 128, CV_RGB(cvRandInt(&rng) % 255, 0, 0),
                    10, 8, 0);
                cvCircle(image, cvPoint(image->width / 2, image->height / 2),
                    image->width / 16, CV_RGB(0, 0, cvRandInt(&rng) % 255),
                    2, 8, 0);
                cvLine(image, cvPoint(0, image->height),
                    cvPoint(image->width / 2, 0), CV_RGB(cvRandInt(&rng)
%255,
                    0, 0), 3, 8, 0);
                cvLine(image, cvPoint(image->width, image->height),
                    cvPoint(image->width / 2, 0), CV_RGB(cvRandInt(&rng)
%255,
                    0, 0), 3, 8, 0);
                cvLine(image, cvPoint(0, image->height),
                    cvPoint(image->width, image->height),
                    CV_RGB(cvRandInt(&rng) % 255, 0, 0), 5, 8, 0);
        }
        times(&finish_processing_time);
```

```
        cvShowImage("Illuminati", image);
        times(&finish_output_time);
        total_output_time += finish_output_time.tms_utime -
            finish_processing_time.tms_utime;
        char c = cvWaitKey(33);
        if(c == 27)
            break;
        i = (i + 1) % (frame_delay + 1);
        total_processing_time += finish_processing_time.tms_utime
            - start_processing_time.tms_utime;
        //вычисление количества кадров в секунду
        printf("%lf fps.\n", (double)frames_amount / ((double)time(NULL)
            - (double)execute_time_start));
    }
    cvReleaseCapture(&capture);
    cvDestroyWindow("Illuminati");
    times(&finish);
    double total_process_time = finish.tms_utime - start.tms_utime;
    printf("Total process time: %lf sec.\n", total_process_time /
clocks_per_sec);
    printf("Video input time part: %lf%%\n", total_input_time
        / total_process_time * 100);
    printf("Video processing time part: %lf%%\n", total_processing_time
        / total_process_time * 100);
    printf("Video output time part: %lf%%\n", total_output_time
        / total_process_time * 100);
    return 0;
}
```

Команда компиляции: gcc -O2 camera.c -o camera -lopencv_core -lopencv_highgui

4. Для оценки скорости обработки видео (количество кадров в секунду) использовался таймер системного времени time(): перед входом в цикл с помощью time() было получено текущее системное время, после чего, перед каждой следующей итерацией цикла, текущее количество кадров делилось на разницу между текущим временем и временем перед входом в цикл.

   Оценка количества кадров в секунду при задержке 33 мс: ≈ 14 fps.

5. Для оценки доли времени, затрачиваемого процессором на обработку (ввод, преобразование, показ) видеоданных, получаемых с камеры использовался таймер времени процесса times(): было измерено общее время работы процесса и время работы конструкций тела цикла, связанных с получением, преобразованием и обработкой видеоданных, после чего вторые величины были поделены на первую, а полученные значение домножены на 100.

   Доля времени, затрачиваемого процессором на обработку (ввод, преобразование, показ) видеоданных:

   1. Ввод изображения: ≈ 4.5%;
   2. Преобразование изображения: ≈ 91.3%;
   3. Вывод изображения: ≈ 1.6%.

3. ВЫВОДЫ:

1. Ознакомились с программированием периферийных устройств на примере ввода данных с Web-камеры с использованием библиотеки OpenCV;
2. Оценили скорость обработки видео (количество кадров в секунду);
3. Оценили долю времени, затрачиваемого процессором на обработку (ввод, преобразование и показ) видеоданных;
4. Ввод и вывод видеоданных не тратят много процессорного времени в сравнении с их обработкой;

5. Простейшие преобразования видеоданных не сильно влияют на количество кадров в секунду.