**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, 23209 группы
**Инокова Семёна Шухратовича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
А. Ю. Кудинов

Новосибирск 2024

# СОДЕРЖАНИЕ

## ЦЕЛЬ

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

## ЗАДАНИЕ

1. Реализовать программу с использованием OpenCV, которая получает поток видеоданных с камеры и выводит его на экран.

2. Выполнить произвольное преобразование изображения.

3. Измерить количество кадров, обрабатываемое программой в секунду. Оценить долю времени, затрачиваемого процессором на обработку (ввод, преобразование, показ) видеоданных, получаемых с камеры.

# ОПИСАНИЕ РАБОТЫ

1. Написал программу на языке C++ с использованием библиотеки OpenCV, позволяющую производить следующие преобразования:
   - Изменение яркости
   - Обнаружение лиц и наложение на них масок
2. Сделал вывод fps в угол окна с частотой обновления 250 мс.
3. Сделал вывод доли времени (в процентах), затрачиваемого процессором на обработку (ввод, преобразование, показ) с частотой обновления 250 мс.

# ЗАКЛЮЧЕНИЕ

*В ходе данной работы я познакомился с библиотекой OpenCV, написал программу, которая делает несколько преобразований изображения, выводит fps и долю затрачиваемого времени. В среднем, если убрать преобразование по распознаванию лиц и наложению масок, fps равен около 27. Распознавание лиц занимает много времени, в связи с чем fps при этом преобразовании значительно падает.*

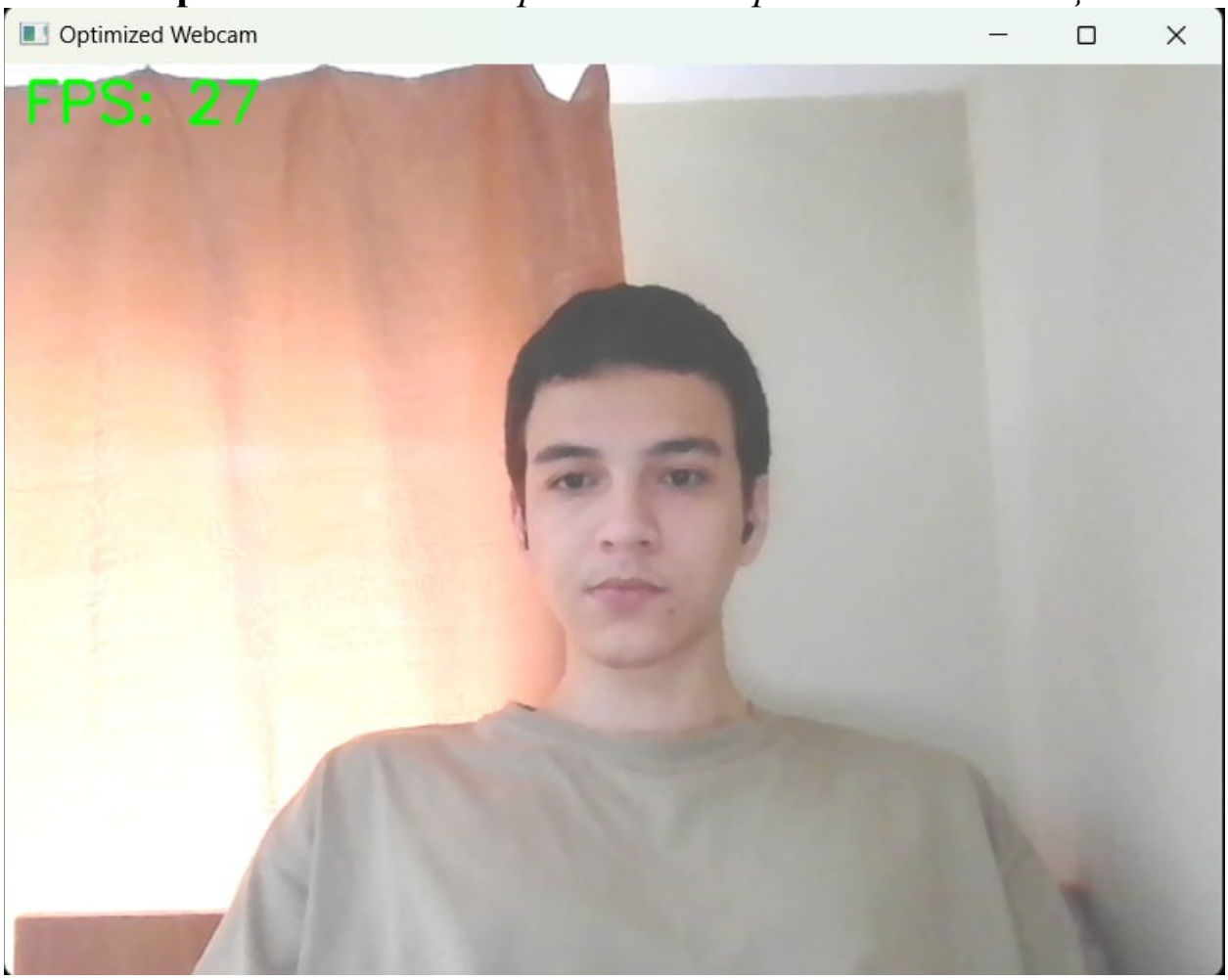
## **Приложение 1.** *Код программы для работы*

```
#include <iostream>
#include <opencv2/highgui.hpp>
#include <opencv2/videoio.hpp>
#include <opencv2/opencv.hpp>
#include <chrono>

void applyOverlay(const cv::Mat& src, cv::Mat& dst, const cv::Mat& overlay, const
cv::Rect& region) {
    cv::Mat resizedOverlay;
    cv::resize(overlay, resizedOverlay, cv::Size(region.width, region.height));

    if (resizedOverlay.channels() == 4) {
        cv::Mat roi = dst(region);

        // Разделение на каналы
        std::vector<cv::Mat> overlayChannels(4);
        cv::split(resizedOverlay, overlayChannels);

        // Альфа-канал (маска прозрачности)
        cv::Mat alpha = overlayChannels[3];
        cv::Mat invAlpha;
        cv::bitwise_not(alpha, invAlpha);

        // Создаем копии цветовых каналов ROI
        std::vector<cv::Mat> dstChannels(3);
        cv::split(roi, dstChannels);

        // Применяем альфа-композитинг для каждого канала
        for (int c = 0; c < 3; ++c) {
            dstChannels[c] = (dstChannels[c].mul(invAlpha, 1.0 / 255.0) +
overlayChannels[c].mul(alpha, 1.0 / 255.0));
        }

        // Собираем итоговый результат
        cv::merge(dstChannels, roi);
    }
}

int main() {
    cv::CascadeClassifier faceCascade;
    if (!
faceCascade.load("C:/opencv/sources/data/haarcascades/haarcascade_frontalface_defaul
t.xml")) {
        std::cerr << "Ошибка: Невозможно загрузить классификатор лиц!" << std::endl;
        return -1;
    }

    cv::Mat overlayImage = cv::imread("C:/EVM/circle.png", cv::IMREAD_UNCHANGED);
    if (overlayImage.empty()) {
        std::cerr << "Ошибка: Невозможно загрузить изображение для наложения!" <<
std::endl;
        return -1;
    }

    cv::VideoCapture capture(0);
    if (!capture.isOpened()) {
        std::cerr << "Ошибка: Невозможно открыть камеру!" << std::endl;
        return -1;
    }

    // Настройка камеры
    capture.set(cv::CAP_PROP_FRAME_WIDTH, 640);
```

```
    capture.set(cv::CAP_PROP_FRAME_HEIGHT, 480);
    capture.set(cv::CAP_PROP_FPS, 30);

    cv::Mat frame;
    const std::string windowName = "Optimized Webcam";
    cv::namedWindow(windowName);

    // Параметры производительности
    double totalReadingTime = 0.0, totalProcessingTime = 0.0, totalOutputTime = 0.0;
    auto startTime = std::chrono::high_resolution_clock::now();
    int frameCounter = 0;
    double fps = 0.0;

    while (true) {
        // Захват кадра
        auto captureStart = std::chrono::high_resolution_clock::now();
        capture >> frame;
        if (frame.empty()) break;
        auto captureEnd = std::chrono::high_resolution_clock::now();
        totalReadingTime +=
std::chrono::duration_cast<std::chrono::milliseconds>(captureEnd -
captureStart).count();

        // Обработка кадра
        auto processingStart = std::chrono::high_resolution_clock::now();

        // Конвертация в черно-белый формат
        cv::Mat grayFrame;
        cv::cvtColor(frame, grayFrame, cv::COLOR_BGR2GRAY);

        // Обнаружение лиц
        std::vector<cv::Rect> faces;
        faceCascade.detectMultiScale(grayFrame, faces, 1.1, 5, 0, cv::Size(80, 80));

        // Наложение маски
        for (const auto& face : faces) {
            applyOverlay(frame, frame, overlayImage, face);
        }

        // Увеличение яркости
        cv::add(frame, cv::Scalar(70, 70, 70), frame);

        auto processingEnd = std::chrono::high_resolution_clock::now();
        totalProcessingTime +=
std::chrono::duration_cast<std::chrono::milliseconds>(processingEnd -
processingStart).count();

        // FPS
        frameCounter++;
        auto currentTime = std::chrono::high_resolution_clock::now();
        double elapsedTime =
std::chrono::duration_cast<std::chrono::milliseconds>(currentTime -
startTime).count();

        std::string fpsText = "FPS: " + std::to_string(static_cast<int>(fps));
        cv::putText(frame, fpsText, cv::Point(10, 30), cv::FONT_HERSHEY_SIMPLEX,
1.0, cv::Scalar(0, 255, 0), 2);

        // Показ кадра
        auto displayStart = std::chrono::high_resolution_clock::now();
        cv::imshow(windowName, frame);
        char c = (char)cv::waitKey(33);
        if (c == 27) break; // Выход при нажатии ESC

        auto displayEnd = std::chrono::high_resolution_clock::now();
```

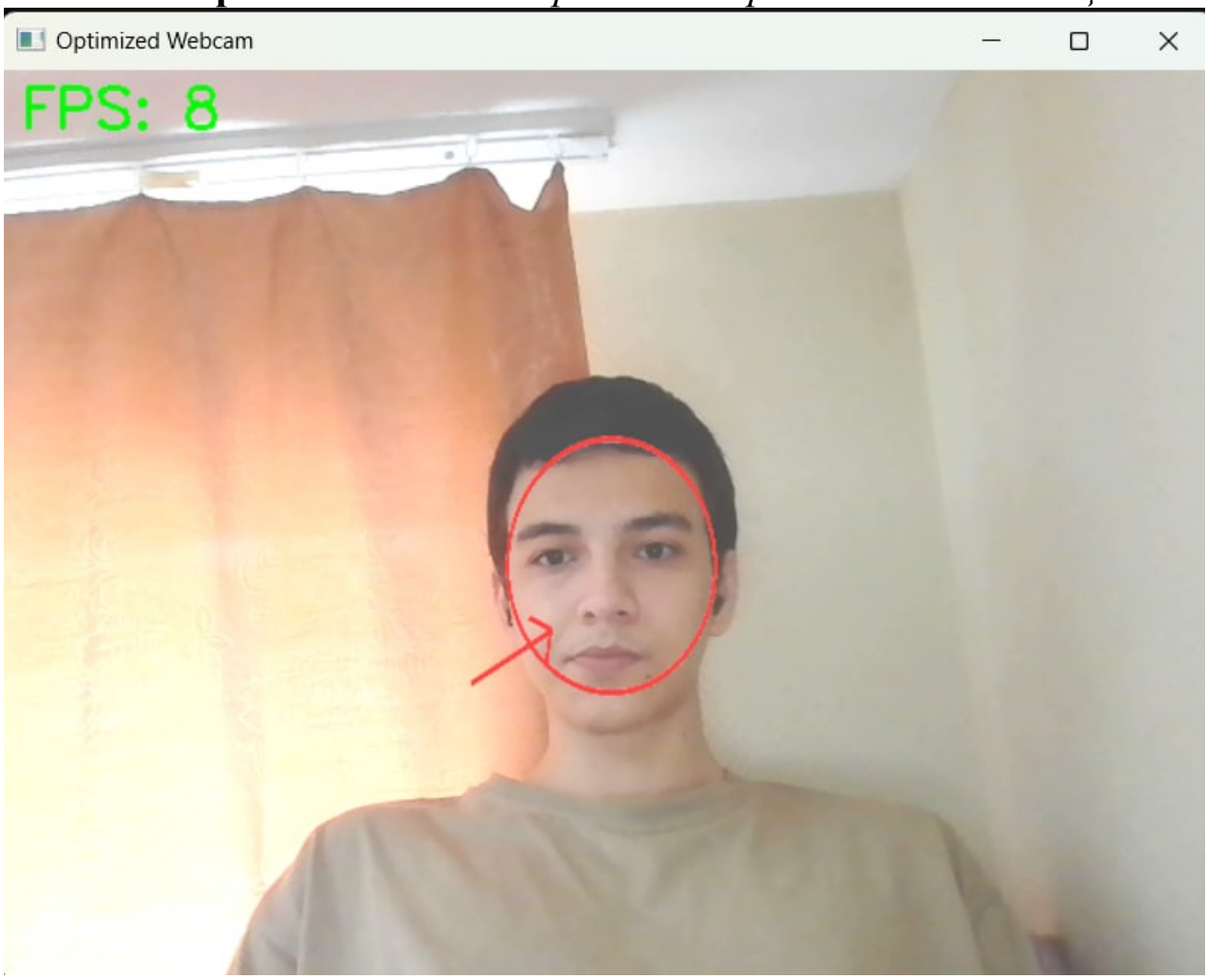
```
        totalOutputTime +=
std::chrono::duration_cast<std::chrono::milliseconds>(displayEnd -
displayStart).count();

        if (elapsedTime > 250.0) {
            fps = frameCounter * 1000.0 / elapsedTime;
            frameCounter = 0;
            startTime = currentTime;

            double totalTime = totalReadingTime + totalProcessingTime +
totalOutputTime;
            std::cout << "Time for processing frames: " << (totalProcessingTime /
totalTime) * 100 << "%" << std::endl;
            std::cout << "Time for reading frames: " << (totalReadingTime /
totalTime) * 100 << "%" << std::endl;
            std::cout << "Time for output frames: " << (totalOutputTime / totalTime)
* 100 << "%" << std::endl;
            std::cout << fps << std::endl;
        }

    }
    capture.release();
    cv::destroyAllWindows();

    return 0;
}
```

**Приложение 2.** *Изображение без распознавания лиц*




Time for processing frames: 0.269028%
Time for reading frames: 3.15548%
Time for output frames: 96.5755%
fps: 27.451

**Приложение 3.** *Изображение с распознаванием лиц*




Time for processing frames: 66.5799%
Time for reading frames: 7.28929%
Time for output frames: 26.1308%
fps: 8.28729