Нижегородский государственный университет им. Н. И. Лобачевского
Лаборатория ITLab

Мини-проект
«Face detection and swap»

Кураторы:
Андрей Петров,
Анна Кривицкая

Участники:
Александр Суслов,
Алексей Чернигин,
Даниил Скатов,
Дмитрий Козлов,
Надежда Дуничкина,
Сергей Ливерко

Нижний Новгород
2006г.

Содержание:

1. Краткий обзор библиотеки IPP

Коммерческая библиотека Intel Integrated Performance Primitives (Intel IPP) представляет собой готовое программное решение для разработки мультимедийных и коммуникационных приложений для вычислительных платформ Intel. Библиотека предлагает разработчикам следующие базовые возможности:

- работа с современными форматами аудио- и видеоданных,
- работа с различными форматами цифровых изображений,
- обработка сигналов,
- обработка речи,
- использование современных криптографических алгоритмов,
- создание ПО, связанного с распознаванием образов,
- создание приложений, связанных с компьютерным зрением,
- обработка матричных и векторных данных,
- обработка строковых данных.

Ключевой особенностью библиотеки Intel IPP является ее оптимизация под современные вычислительные платформы Intel. Она ориентирована для оптимальной работы как на процессорах Intel для рабочих станций, мобильных и наладонных компьютеров, так и на серверных решениях от Intel.

Для выполнения данной части проекта разработчикам понадобилась часть библиотеки IPP, отвечающая за обработку цифровых изображений. Как уже было отмечено ранее, библиотека OpenCV, также задействованная в данном проекте, может работать в режиме использования IPP. Таким образом, библиотека IPP неявно используется в других частях проекта. В этой главе на примере конкретной задачи мы покажем, как можно использовать библиотеку IPP явным образом.

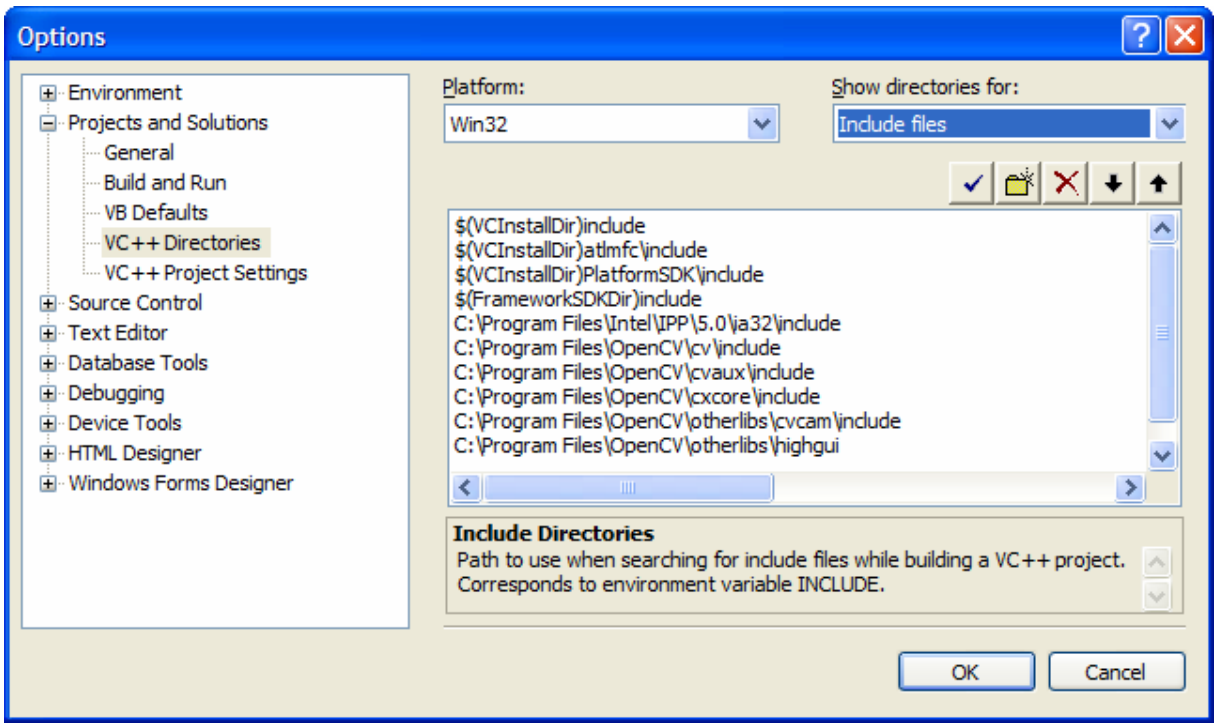
Конечно, использование IPP для обработки цифровых изображений имеет свои особенности. При решении этой задачи наиболее популярны два подхода. Первый из них (если разработка ведется для ОС Windows) состоит в использовании стандартных средств интерфейса GDI через Win 32 API. Очевидно, что набор этих средств весьма ограничен, и конечное решение оказывается непроизводительным. Второй подход заключаается в использовании библиотек сторонних производителей. Эти библиотеки имеют различное внутреннее устройство, могут быть написаны как на языке высокого уровня, так и с привлечением инструкций языка ассемблера. Использование IPP в задачах обработки изображений, как нам кажется, облает преимуществами двух этих подходов, сводя к минимуму их недостатки. Действительно, эта библиотека оптимизирована для всех современных платформ Intel, что позволяет добиться максимальной производительности при ее использовании. В то же

время она реализована как для Windows, так и для Linux, что делает продукт, разработанный с использованием IPP, легко переносимым между платформами.

2. Подготовка к работе с IPP

Покажем далее, какие действия нужно произвести, чтобы начать работу с библиотекой IPP. В первую очередь необходимо зарегистрироваться на сайте www.intel.com, получив тем самым возможность скачать дистрибутив IPP. По адресу электронной почты, указанному при регистрации, будет выслан файл, содержащий сведения о лицензии. При установке библиотеки пользователь должен указать путь, по которому инсталлятор сможет найти этот файл.

Установив библиотеку, нужно настроить среду Microsoft Visual Studio .NET.



Запустив среду, выберем пункт меню «Tools» – «Options», и в списке слева найдем элемент «VC++ Directories». Предположим, что установка библиотеки IPP была произведена с параметрами по умолчанию. Тогда в соответствующие списки необходимо добавить следующие директории:

- ***Include files:***

  C:\Program Files\Intel\IPP\5.0\ia32\include;

- ***Library files:***

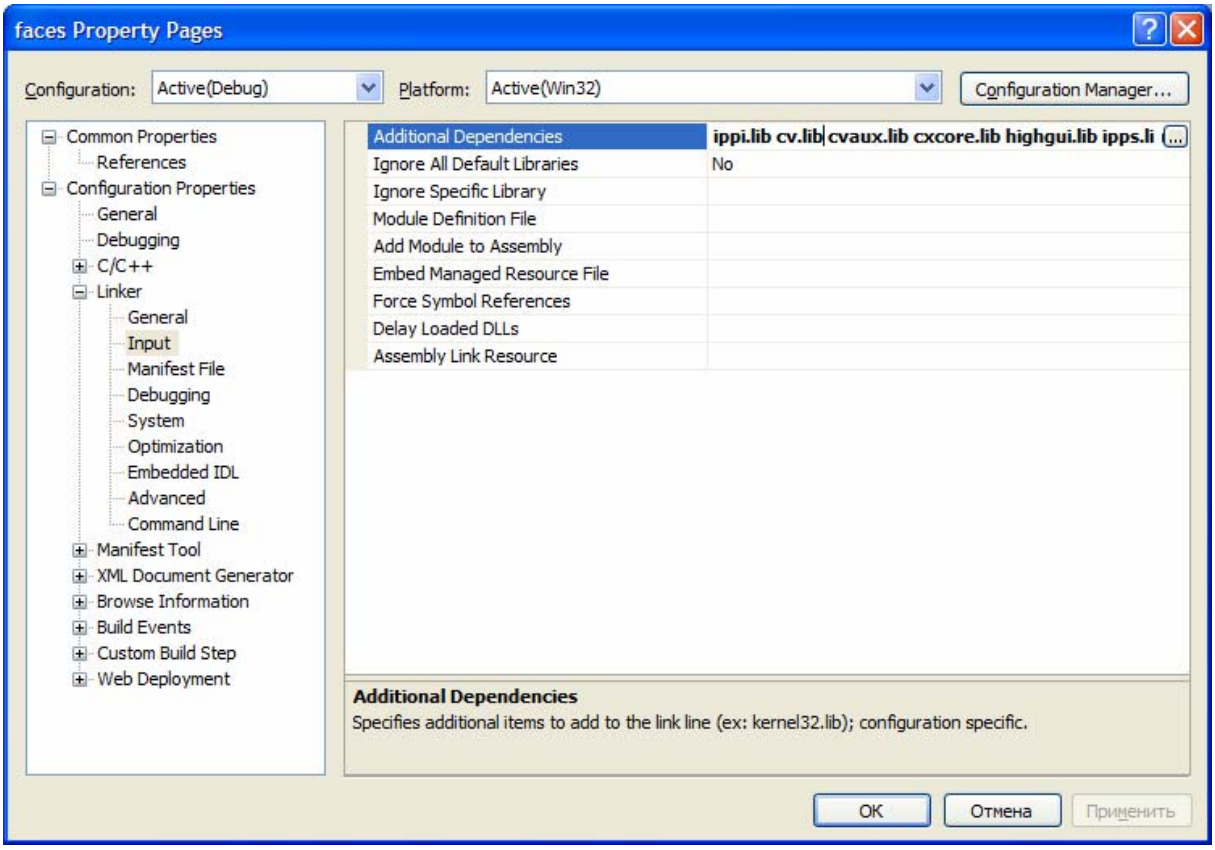
  C:\Program Files\Intel\IPP\5.0\ia32\lib,

  C:\Program Files\Intel\IPP\5.0\ia32\stublib.

Пусть создан некоторый проект C++. Чтобы в нем можно было использовать возможности библиотеки IPP по работе с изображениями, необходимо в свойствах проекта (соответствующее окно вызывается из меню «Project») найти в списке слева элемент «Linker — In-

put», и к текстовому полю «Additional dependencies» добавить имя библиотеки IPP, которая нужна для работы: «ippi.lib» (для нашего случая).

Теперь проект настроен. Для использования функций IPP достаточно подключить к файлу с исходным текстом заголовочный файл ***ipp.h***.



Далее мы изложим те аспекты использования IPP, которые либо могут быть упущены или не поняты при изучении официальной документации, либо отсутствуют в этой документации. Знание изложенного ниже материала необходимо для того, чтобы в ходе дальнейшего изложения понимать, как работает программа.

3. Особенности использования IPP

3.1. Соглашения об именах функций IPP

При использовании библиотек важным является вопрос о соглашениях для имен функций. Есть свои соглашения и в библиотеке IPP. Как правило, в сигнатуру любой функции IPP входит объект-преемник, объект-источник, плюс некоторые параметры выполняемой операции. В нашем случае объектами чаще всего являются изображения. Под прямоугольным изображениям понимается определенным образом упорядоченный набор элементов определенного типа данных, непрерывно расположенных в памяти.

Все типы и функции библиотеки поделены на домены, каждый из которых отвечает за определенную предметную область и имеет уникальное короткое наименование. Нас инте-

ресует домен ***ipps***, предназначенный для обработки цифровых изображений. Имя любой функции IPP начинается с имени домена, которому она принадлежит.

Далее следует имя самой функции, возможно, с большой «C» в конце — это означает, что одним из операндов функции является константа. После имени ставится символ подчеркивания, и записан тип данных и размер данных, с которыми оперирует функция. Эта информация записывается следующим образом: `<bits>|<u|s|f>[c]`. Bits обозначает количество битов в одном элементе данных. `<u|s|f>` — можно выбирать типы `unsigned integer`, `signed integer`, `float`. `[c]` — указывает, что элемент данных комплексный, т. е. он представляет собой два упорядоченных элемента указанного типа. Комплексные величины часто используются при создании мультимедийных приложений и игр — например, для реализации преобразования поворота.

Далее следует символ подчеркивания, после которого может стоять один или несколько суффиксов. Для каждого домена существует свой набор суффиксов. Мы будем рассматривать суффиксы для домена ***ipps***.

3.2. Способ представления изображений

Как известно, растровые изображения можно представлять по-разному. Причем, различными могут быть как *способ представления*, так и *формат хранения изображений*. Способ представления изображения определяет то, как интерпретируется формат хранения, поэтому начинать следует с него. Определяющим понятием при описании представления является понятие *палитры* — набора цветов, который учувствует в формировании конечного цвета каждого пикселя изображения. Для описания каждого пикселя изображения нужно задать *интенсивность* каждого цвета, входящего в палитру. В данной программе используется наиболее популярная палитра *RGB*, т.е. каждый пиксель задается интенсивностями красного, зеленого и синего цветов.

Формат хранения определяет и такой параметр, как *качество изображения*, которое зависит от размеров памяти, выделяемой под каждый пиксель изображения. Чем больше эта память, тем больший диапазон цветов может использоваться в изображении. Размер памяти, выделяемой под каждый пиксель, называется *глубиной цвета*. Программа в силу некоторых особенностей библиотеки компьютерного зрения OpenCV работает только с изображениями с глубиной цвета 24 бита (по 8 бит под красный, зеленый и синий цвета). Это ограничение не является критичным, так как глубины цвета в 24 бита вполне достаточно для представления качественных картинок без альфа-канала. Функциям, работающим с изображениями такого типа, в IPP соответствует дескриптор `8u_C3`.

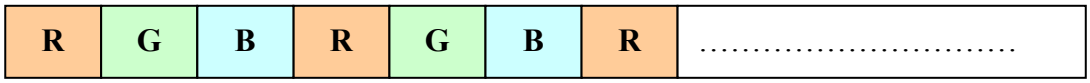
Альфа-канал, иногда еще ошибочно называемый «прозрачностью», представляет собой дополнительный компонент для каждого пикселя изображения. Принято присутствие альфа-канала в изображении обозначать в палитре, поэтому говорят, что изображение представлено в палитре RGBA, где A как раз и означает наличие альфа-канала. Глубина цвета при этом, как видно, увеличивается до 32 бит (если предположить, что под все компоненты выделяется по 8 бит). При обычном просмотре изображения — например, в программе просмотра изображений, альфа-канал не отображается, т.е. изображение в RGB выглядит так же, как и RGBA. Дело в том, что альфа-канал используется для создания специальных

эффектов в таких графических редакторах как Adobe Photoshop или The GIMP. Так же применяется альфа-канал в играх, в хороших видео-редакторах при использовании динамических масок, т.е. везде, где используется наложение и комбинирование. Наиболее используемым эффектом альфа-канала является прозрачность – это и объясняет его второе (и ошибочное) название. Дело в том, что альфа-канал сам по себе представляет просто числовое значение, но если при задании, например, RGB палитры мы оговариваем, что R значит интенсивность красного, G – интенсивность зеленого, B - интенсивность синего цветов, то при задании альфа-канала мы не оговариваем, как будет использоваться его значение. Интерпретация этого значения полностью ложится на программу, работающую с изображением, и следовательно от нее зависит, что будет «представлять» альфа-канал в данном изображении. Так или иначе, альфа-канал очень удобное и гибкое средство для комбинирования изображений. О его использовании в данной программе читайте ниже.

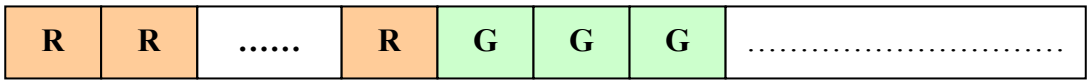
3.3. Формат хранения изображений

Теперь необходимо сказать о том, как хранится изображение в памяти. Существует два наиболее распространенных подхода:

- ***Pixel-oriented approach*** (*пиксельно-ориентированный подход*). При использовании данного способа хранения изображение хранится в памяти последовательно пиксель за пикселем, что можно проиллюстрировать так:



- ***Planar-oriented approach*** (*плоскостно-ориентированный подход*). При этом подходе изображение представляется при помощи цветовых плоскостей. Т.е. сначала идут красные компоненты для всех пикселей, затем зеленые и т.д.



Последовательность красных компонент называется красной *цветовой плоскостью*, аналогично остальные компоненты.

Программа использует для представления изображений первый подход.

Остается еще обсудить некоторые нюансы хранения изображений. Логично представлять себе изображение как двумерный массив — ведь картинка предстает перед нами именно в двумерном виде. Ничего сложного в этом нет, ведь, как известно двумерный массив хранится в памяти последовательно, и никакого противоречия с предыдущими рассуждениями нет, но здесь надо учитывать один тонкий момент — *выравнивание*. Дело в том, в силу особенностей архитектуры x86 процессор гораздо быстрее обращается к адресам памяти, выровненным на границу двойного слова (4 байта). Поэтому размер горизонтальной строки изображения округляется в большую сторону для достижения этого выравнивания, и остаток (от реального размера строки до выровненного) заполняется нулями. При этом становится необходимым такой параметр изображения, как шаг (step) — расстояние в байтах между начальными байтами последовательных строк. Шаг должен в таком случае ис-

пользоваться для адресации вместо горизонтального размера картинки. Его использование объясняется ниже.

Как правило, пользователю библиотеки IPP не приходится вручную вычислять значения шага. Эти значения передает сама IPP при создании и обработке изображений Чаще всего достаточно использовать отдельную переменную для хранения шага и передавать ее значение в качестве параметров соответствующих функций IPP.

3.4. Regions of interest (активные области изображения)

Функции IPP позволяют работать не только с целым изображением, но и с некоторой его прямоугольной частью. *Активной областью изображения* (*image region of interest*, или *ROI*) называют прямоугольную область, которая может быть как частью изображения, так и охватывать все изображение. С помощью использования ROI можно добиться прироста производительности, поскольку отпадает необходимость выделения отдельного буфера и перемещения в него части изображения для дальнейшей обработки. Функции IPP, предполагающие работу с ROI, имеют в конце своего имени суффикс `R`.

ROI полностью определен, если для него указаны количество пикселей и координаты верхнего левого угла `(RoiOffset.x, RoiOffset.y)`. Эта точка задается в системе координат, связанной с верхним левым углом изображения. Система координат, связанная с изображением, устроена обычным образом: ось абсцисс направлена из этой точки влево, ось ординат — вниз. Отметим, что в общем случае функция требует задания как ROI изображения-источника, так и ROI изображения-преемника. Если это так, то количество пикселей в ROI для источника и для преемника считается одинаковым.

Функции IPP, работающие с ROI, требуют в качестве обязательного параметра указатель на начало ROI (для источника `pSrc`, для преемника `pDst`). Вспомним, как в IPP устроено изображение (см. выше) и дадим формулу для нахождения этого указателя (`pROI`). Пусть обрабатывается `n`-канальное изображение с шириной `img.width` и высотой `img.height`, расположенное по указателю `pImg`. При работе с функциями IPP в качестве параметров им необходимо передавать значения `srcStep` и `dstStep`, представляющие собой значения шагов для изображения-источника и изображения-преемника.

```
pROI = pImg + n * (img.step * RoiOffset.y) + RoiOffset.x.
```

4. Алгоритм

Общая схема алгоритма перестановки лиц состоит в следующей последовательности.

- Получение кадра (`img`) и кадра с контурами (`src`)
- Получение дополнительных данных
- Расчет размеров массивов
- Выделение регионов с лицами в отдельные массивы (`fcr[i]`)
- Масштабирование и вращение

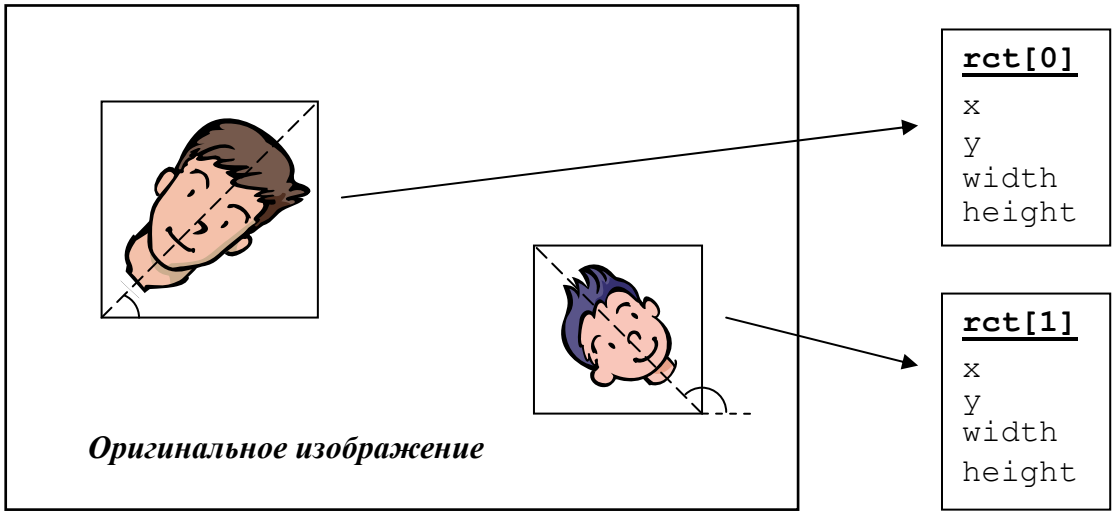
- Создание масок (`mskb[i]`, `mskt[i]`)
- Расчет координат для наложения
- Наложение
- Улучшение визуализации

Далее подробно рассмотрим каждый шаг алгоритма.

4.1. Получение кадра с контурами и дополнительных данных

В рамках данных пунктов производится захват кадра из видео-потока (с веб-камеры, либо из готового видео-ролика) или загрузка статической картинки. Полученное изображение обрабатывается функцией локализации лиц. Функция локализации лиц производит следующие действия:

1) Нахождение прямоугольников, заключающих лица(`rct[i]`)
2) Нахождение приблизительных контуров лиц в данных прямоугольниках
3) Нахождение углов поворота лиц относительно горизонтали (вертикали)



Найденные функцией прямоугольники (представленные как структуры `CvRect`) помещаются в глобальное хранилище, `faces_storage` реализованное при помощи объекта из библиотеки OpenCV `CvMemStorage`.

Оригинальное изображение (`img`) копируется во вспомогательный массив (`src`) и в данном массиве происходит выделение контуров. Контуры помечаются чистым зеленым цветом `RGB(00,FF,00)`. Массив `src` передается для дальнейшей обработки.

Так же функцией выполняется нахождение угла поворота лиц относительно горизонтальной (вертикальной оси).

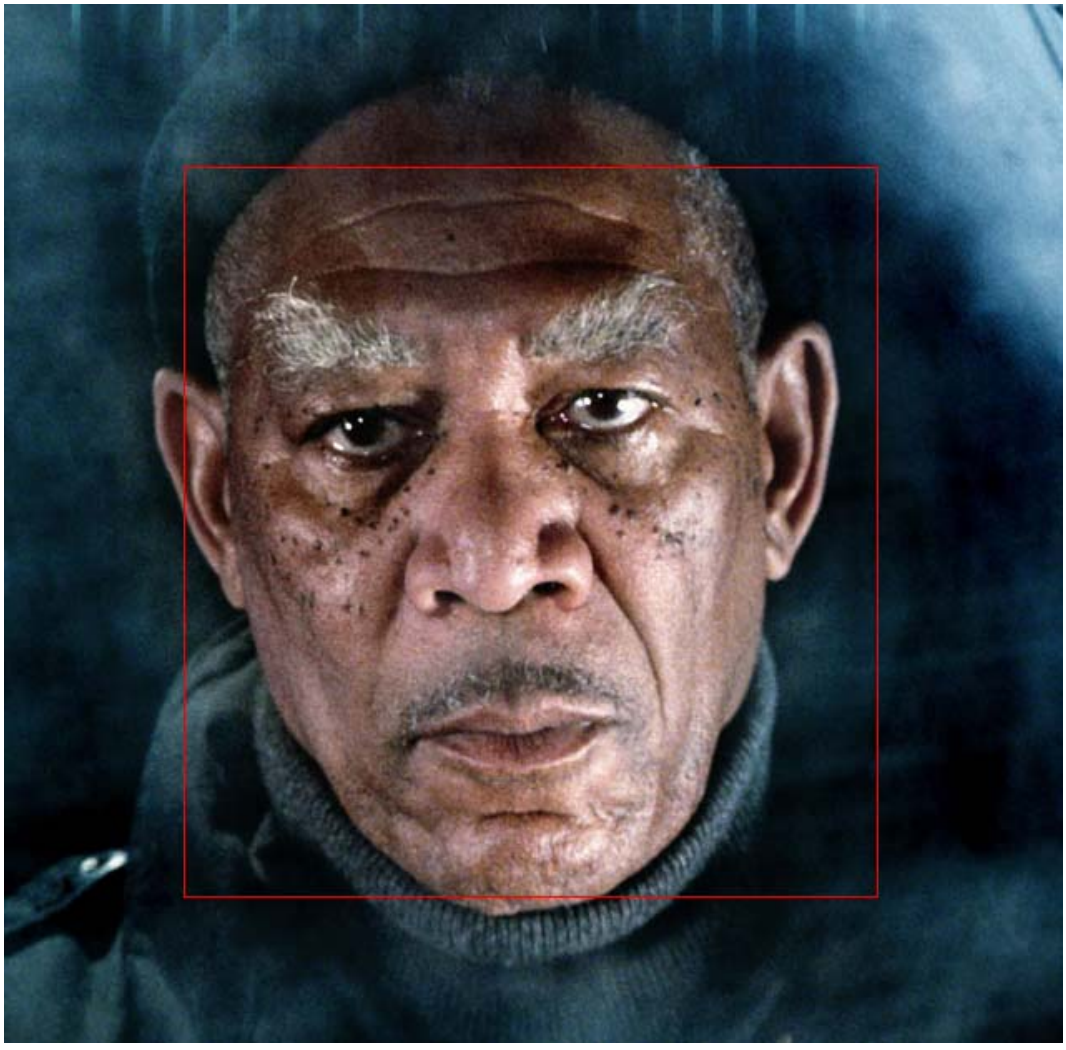
4.2. Локализация лица и определение его контура

Локализация лица

Первым пунктом решения задачи перестановки лиц является локализация лиц на фотографии или в кадре видеопотока, т.е. определение горизонтального прямоугольника, описанного около предполагаемого лица. С этой задачей неплохо справляется функция

```
cvHaarDetectObjects(const CvArr* image,
                    CvHaarClassifierCascade* cascade,
                      CvMemStorage* storage,
                    double scale_factor=1.1,
                      int min_neighbors=3,
                    int flags=0,
                      CvSize min_size=cvSize(0,0) )
```

библиотеки **OpenCV**, которая возвращает последовательность всех найденных лиц.



Таким образом, мы имеем квадратную область, в которую, по предположению, вписано лицо.

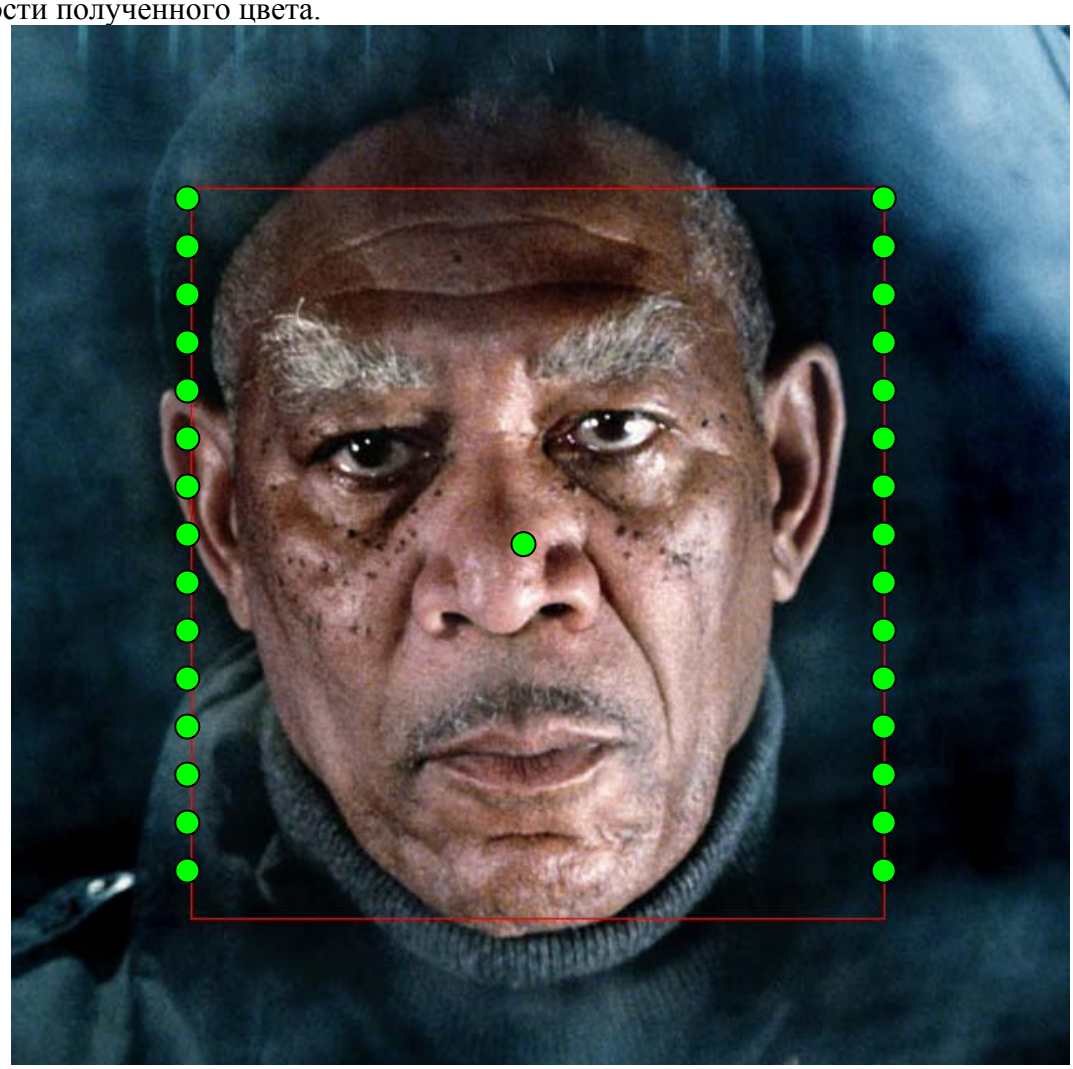
Определение контура лица

Следующим этапом перестановки лиц является определение контура лица в том регионе, в котором, по предположению, содержится искомое абстрактное лицо. Но задача состоит не в том, чтобы абсолютно точно определить контур лица, а лишь в том, чтобы как можно точнее приблизить контур некоторой ломаной (чтобы при перестановке лиц на новое место попало как можно меньше посторонних пикселей). Другими словами, надо отсечь от заданного прямоугольника всё (по возможности), что не относится к лицу.

Для решения поставленной задачи предлагается следующий алгоритм.

1) На левом и правом ребре полученного квадрата расставляем через некоторый промежуток (который определяется экспериментально) “щупы”, которые, впоследствии, будем двигать.

2) Так как лицо, по предположению, вписано, то середина квадрата будет обязательно принадлежать лицу. Возьмём в центре квадрата и в некоторой окрестности центра пробы цвета и найдём некоторый средний цвет **`Col`**, как среднее арифметическое соответствующих компонент взятых на пробу цветов. Цвет кожи лица будет находиться в некоторой окрестности полученного цвета.



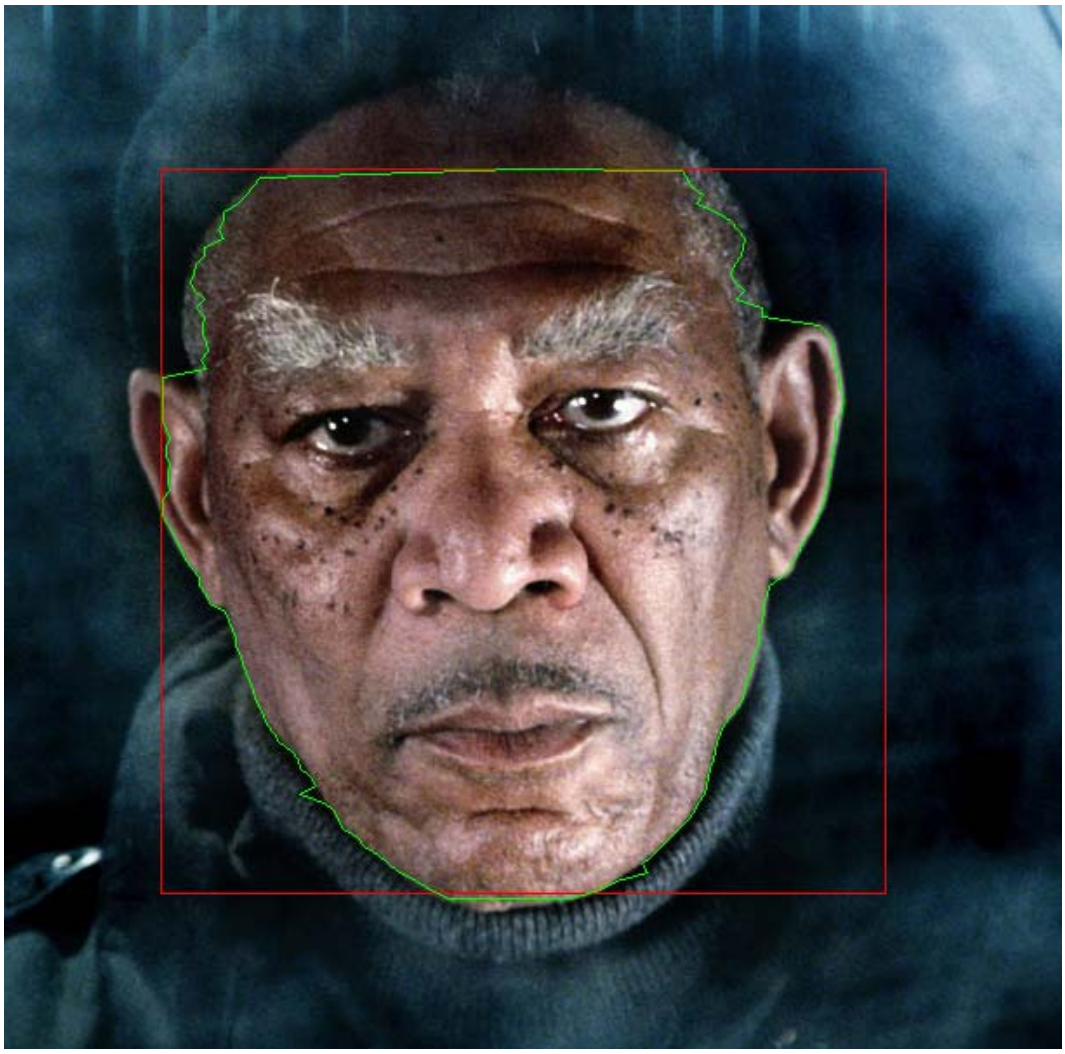
3) Необходимо определить искомый цветовой диапазон. Пусть **`Cur`** – цвет пикселя в текущем положении какого-то щупа. Обозначим

$$k1 = \frac{Cur.R}{Col.R},\ k2 = \frac{Cur.G}{Col.G},\ k3 = \frac{Cur.B}{Col.B}$$

Если будут выполняться условия

$$\begin{cases} \left|\frac{k1}{k2} - \frac{k2}{k3}\right| < \varepsilon, \\ \left|\frac{k2}{k3} - \frac{k3}{k1}\right| < \varepsilon, \\ \left|\frac{k3}{k1} - \frac{k1}{k2}\right| < \varepsilon, \end{cases}$$

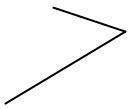
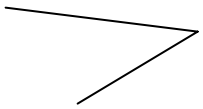
то **Cur ∈ [Col - ε; Col + ε],** т.е. цвет **Cur** принадлежит найденному цветовому диапазону. Таким образом, будем двигать щупы до тех пор, пока не окажемся в пикселе, цвет которого принадлежит найденному диапазону. Соединив щупы в исходном порядке линиями, получим сильно изрезанный контур.



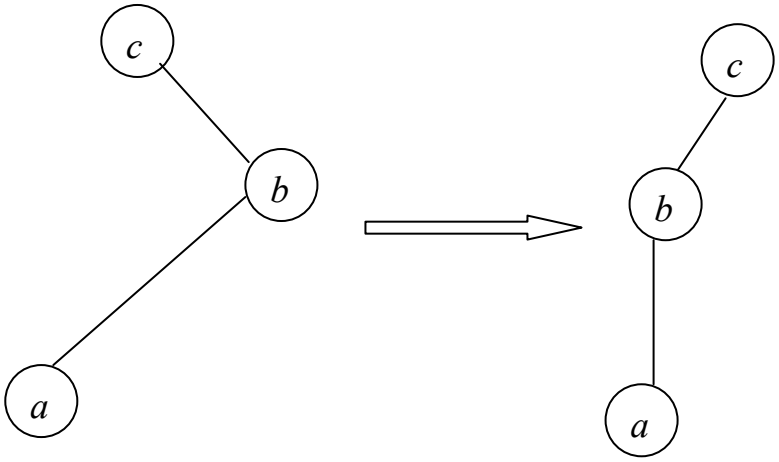
На этой фотографии контур изрезан слабо, но в большинстве случае эффект “пилы” намного сильнее, и поэтому недопустимо переставлять полученную на этом этапе область. Необходимо выровнять контур.

Выравнивание контура

Рассмотрим левую часть контура (для правой части – всё будет аналогично). И отдельно разберём верхнюю половину этого контура (для нижней половины всё будет абсолютно также, с точностью до симметрии). Возможны 2 варианта неправильных изломов изломов (напомним, что наша задача – получить выпуклый многоугольник).

a)  *b*) 

Для того, чтобы эти изломы были частью выпуклого многоугольника, их надо преобразовать следующим образом (как один из вариантов):

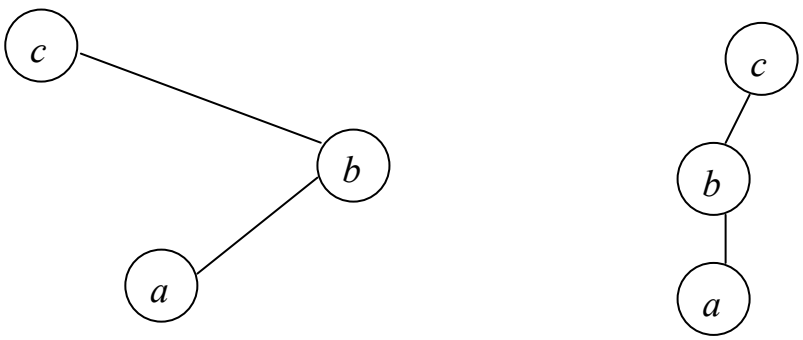
a) 

Очевидно, что если координаты точек текущего излома удовлетворяют следующим условиям

$$\begin{cases} (C.x - B.x) < 0, \\ (A.x - B.x) < 0, \\ C.x > A.x, \end{cases}$$

то текущий излом является изломом первого типа и его можно скорректировать следующим образом:

$$B.x = C.x - \frac{(C.x - A.x)}{2}.$$

b) 

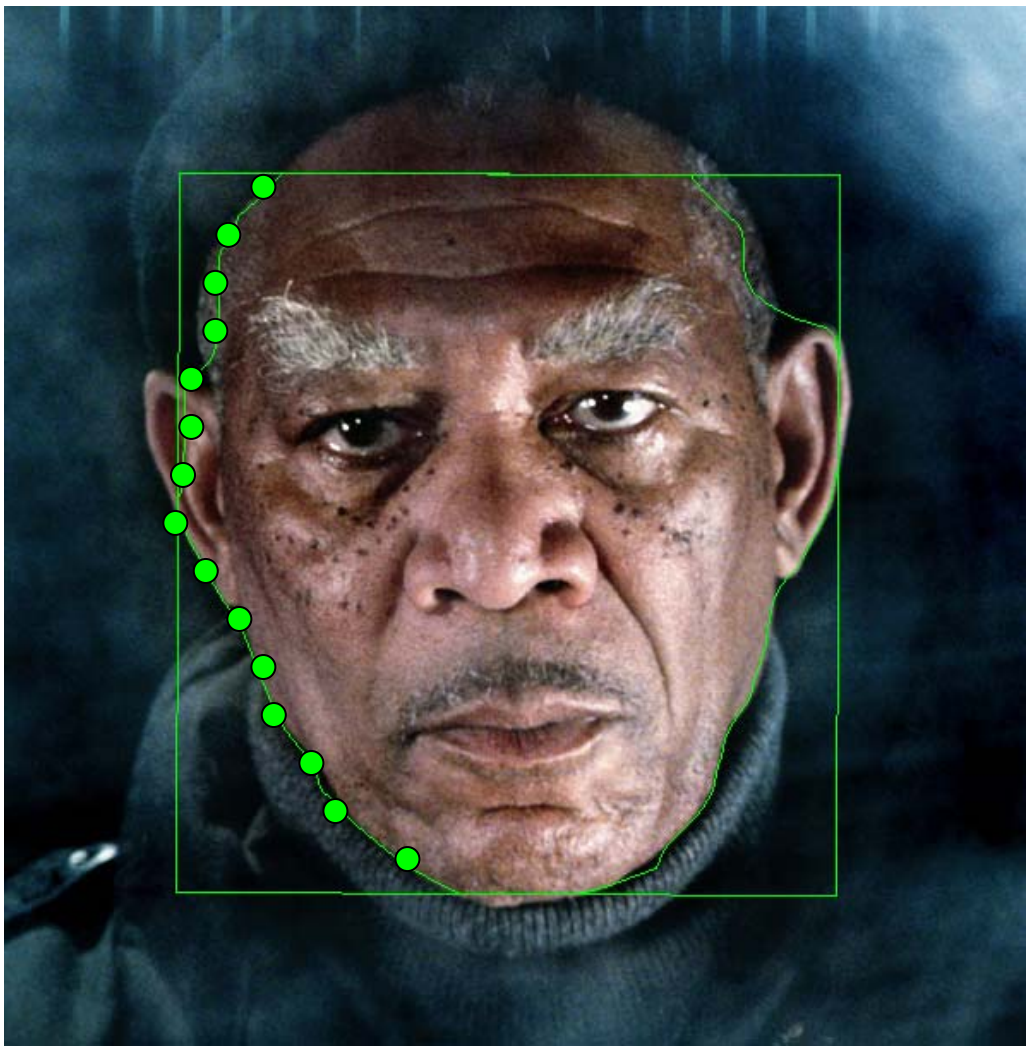
Очевидно, что если координаты точек текущего излома удовлетворяют следующим условиям

$$\begin{cases} (C.x - B.x) < 0, \\ (A.x - B.x) < 0, \\ C.x < A.x, \end{cases}$$

то текущий излом является изломом второго типа и его можно скорректировать следующим образом:

$$\begin{cases} B.x = A.x, \\ C.x = A.x + 1 \end{cases}$$

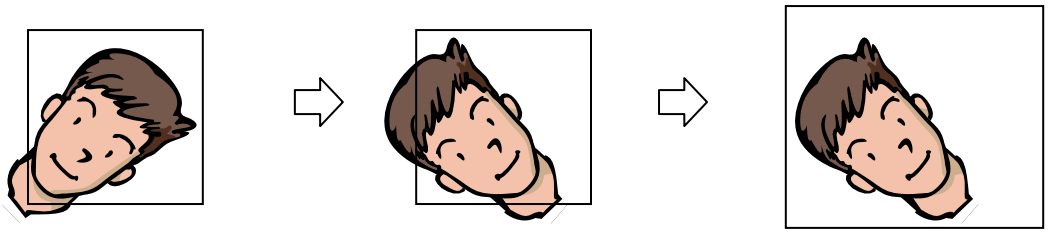
Подкорректировав 2 раза все встретившиеся изломы, мы получим приемлемую картинку.



4.3. Выделение регионов в отдельные массивы

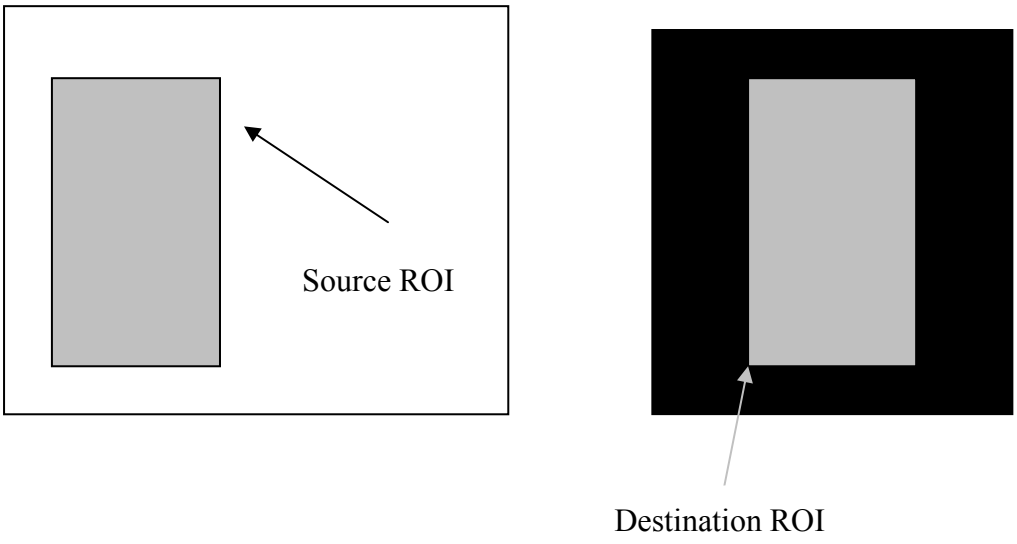
Для продолжения работы необходимо скопировать полученные на предыдущем шаге прямоугольники с лицами в отдельные массивы. Необходимо это для последующей замены лиц. Следует обратить внимание, что размеры данных массивов не совпадают с разме-

рами прямоугольников, содержащих лица. Сделано это для того, чтобы не потерять часть картинки при последующих поворотах и масштабировании изображения (см. рисунок). Массив представляет собой квадратную матрицу, размер которой для *i*-го лица вычисляется как диагональ прямоугольника `rct[i]`, что гарантирует нам отсутствие выхода картинки за пределы квадрата при повороте на любой угол.



Оригинальное изображение ***Потеря части изображения при повороте*** ***Решение проблемы: увеличение размера массива***

Выделение памяти осуществляется динамически при помощи средств библиотеки Intel IPP. Для выделения памяти используется функция `ippiMalloc_8u_C3`. После того, как память выделена, производится копирование прямоугольных фрагментов начального изображения, содержащих лица, в новые массивы. Копирование производится таким образом, чтобы прямоугольник находился в центре нового массива (см. рисунок). Копирование осуществляется функцией `ippiCopy_8u_C3R`, где в качестве ROI в исходном изображении используется прямоугольник с лицом, а роль ROI в новом массиве играет прямоугольник такого же размера в центре. Часть нового изображения, которая не копируется из кадра, заполняется черным цветом.



Таким образом, мы получаем изображения с локализованными лицами, которые пригодятся нам при создании масок.

4.4. Выполнение масштабирования и поворота

В ходе работы программы предварительно были определены углы, на которые повернуты главные оси лиц относительно оси горизонтали: α_1 и α_2 соответственно. При этом, зная размеры прямоугольников, в которые вписаны лица, мы смогли вычислить масштабный коэффициент coeff — он указывает, на сколько первый прямоугольник больше второго. Тогда, чтобы начать обмен лиц, необходимо изображение первого лица уменьшить в coeff раз и повернуть его на угол $\alpha_1 - \alpha_2$ против часовой стрелки, а второго лица — увеличить в coeff раз и повернуть на угол $\alpha_2 - \alpha_1$ против часовой стрелки.

Для выполнения поворота и последующего масштабирования изображений лиц мы используем функции `ippiRotateCenter_8u_C3R` и `ippiResizeCenter_8u_C3R`. Сразу понятно, что эти функции предназначены для преобразования изображения относительно его центра.

Функция `ippiRotateCenter_8u_C3R` имеет следующую сигнатуру:

```
IppStatus ippiRotateCenter_8u_C3R(
   const Ipp8u pSrc, IppiSize srcSize,
   int srcStep,IppiRect srcRoi, Ipp8u* pDst,
   int dstStep, IppiRect dstRoi, double angle,
   double xCenter, double yCenter, int interpolation).
```

В данном случае для srcStep передаем утроенную ширину изображения-источника, для dstStep — утроенную ширину изображения-преемника, используем быструю интерполяцию «по ближайшему соседу» (nearest neighbour), которой соответствует константа `IPPI_INTER_NN`.

Похожие значения мы передаем и функции `ippiResizeCenter_8u_C3R`, параметры которой почти идентичны параметрам `ippiRotateCenter_8u_C3R`. Вместо угла эта функция принимает пару масштабных коэффициентов для длины и ширины изображения.

После поворота на изображении-преемнике, роль которого выполняет заранее подготовленный буфер, остается некоторое пустое пространство вокруг поворачиваемого лица. Буфер мы предварительно заливаем белым цветом, поэтому пространство также белое.

Поворот и масштабирование вынесены в отдельную функцию. Ее сигнатура:

```
void Rotate_image (
   Ipp8u** img,              // изображение-источник
   IppiSize* size,           // размеры изображения-источника
   double angle,             // угол поворота
   Ipp32f coeff)             // коэффициент масштабирования
```

4.5. Завершающая часть алгоритма

Осталось выполнить создание масок (`mskb[i]`, `mskt[i]`), расчет координат для наложения, и собственно наложение. Задача состоит в том, чтобы наложить массив с локализованным лицом на кадр, в то его место, где находится другое лицо, и при этом в кадре ото-

бразиться должна только часть нашего массива, которая заключена в найденном функцией на первом шаге контуре, т.е. предположительно лицо.

Для наложения одного лица на место другого с сохранением фона применяется алгоритм, называемый маскированием. В данном алгоритме участвуют исходное изображение и массивы с локализованными лицами. Хочется отметить, что использование маскирования не ограничивается используемым нами наложением с сохранением фона. Маскирование также применяется для создания различных специальных эффектов. Общая схема применяемого нами маскирования следующая:

- Создание черно-белой маски
- Создание целевой маски
- Комбинирование

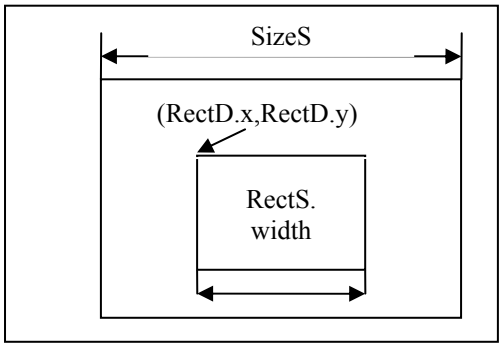
Для понимания алгоритма начать объяснение следует именно с комбинирования. Как говорилось ранее, каждый пиксель изображения задается интенсивностями трех цветов RGB и, возможно, альфа-каналом. Обозначим интенсивности каждого пикселя исходного изображения (на которое мы и будем накладывать) через `(Rd,Gd,Bd,Ad)` и назовем это изображение приемником (`destination`), т.к. накладываем мы именно на него. Интенсивности изображения, которое мы собираемся накладывать, обозначим через `(Rs,Gs,Bs,As)`, изображение назовем источником (`source`).

Для гибкого комбинирования цветов вводятся факторы источника (`Rsf,Gsf,Bsf,Asf`) и приемника (`Rdf,Gdf,Bdf,Adf`), которые мы задаем сами, в зависимости от нужного нам эффекта наложения.

Далее следует определить, в какое место кадра накладывать изображение с лицом, ведь размеры источника гораздо больше размеров приемника, и нужно наложить именно так, чтобы создалась видимость замены лица. Данный расчет выполняется так. Пусть `rectS` — прямоугольник, содержащий лицо, которое мы собираемся переставить, `sizeS` — размер массива с лицом, которое мы собираемся переставить, `rectD` — прямоугольник, содержащий лицо, на место которого мы переставляем. Координаты точки, с которой должен совпасть левый верхний угол нашего массива и масок при наложении, вычисляются:

```
x = rectD.x – (sizes - rectS.width) / 2
y = rectD.y – (sizes - rectS.height) / 2
```

Для понимания этих расчетов достаточно посмотреть на рисунок:



Т.о. мы определили координаты наложения, или, другими словами, мы теперь знаем, какие пиксели источника и приемника комбинировать.

Результирующий цвет пикселей приемника, участвующих в комбинировании, с учетом выбранных нами факторов теперь вычисляется по формуле:

```
(Rs*Rsf ¤ Rd*Rdf, Gs*Gsf ¤ Gd*Gdf,Bs*Bsf ¤ Bd*Bdf,As*Asf ¤ Ad*Adf)
```

Где ¤,* — выбираемые нами операции. Очевидно, что за счет подбора факторов и операций можно достичь очень гибкого комбинирования.

Это общий подход. Теперь следует объяснить, как использовать этот подход для создания требуемого эффекта прозрачности части изображения-источника. Данный эффект достигается за счет масок и двойного комбинирования.

Для первого комбинирования создается черно-белая маска. Часть маски, соответствующая части изображения, которую мы хотим в конечном итоге увидеть в кадре, должна быть заполнена черным цветом, оставшаяся часть – белым.

Выберем в качестве факторов источника и приемника соответственно вектора `(0,0,0,0)` и `(Rs,Gs,Bs,As)`, в качестве операции ¤ выберем сложение, * - побитовое логическое «и». Формула примет вид:

```
(Rs and Rb,Gs and Gd,Bs and Bd,As and Ad)
```

Очевидно, что цвет пикселей приемника под белой частью маски не изменится, черная же часть отобразится в кадр. Далее происходит создание целевой маски. Целевая маска представляет собой изображение того же размера, что и черно-белая маска (и такое же, как изображение с локализованным лицом). В этом изображении, та его часть, что соответствует белой части черно-белой маски, заполняется черным цветом, другая часть (что соответствует черной части черно-белой маски) заполняется пикселями изображения с локализованным лицом.

Второе комбинирование проводится над изображением-приемником и целевой маской с теми же факторами, но операцией * — поразрядное логическое «или». Формула принимает вид (в качестве изображения-приемника выступает результат предыдущего комбинирования):

```
(Rs or Rb,Gs or Gd,Bs or Bd,As or Ad)
```

Т.о. черная часть целевой маски не отобразится в изображении-приемнике, а цветная часть будет отображена.

Побитовые логические операции над изображениями производятся функциями библиотеки IPP: `ippiAnd_8u_C3R` и `ippiOr_8u_C3R`. Для получения других эффектов, таких как размытие, затухание, подсветка и т.д. следует использовать другие комбинации операций, факторов и масок.

4.6. Алгоритмы создания масок

Данная программа использует два подхода к созданию масок:

- Маски на основе контуров
- Маски на основе вписанных эллипсов

Преимуществами первого подхода являются: возможность определения угла поворота лица относительно горизонтальной (вертикальной) оси и теоретически более точное маскирование (т.е. на качественных картинках результат лучше). Преимуществами второго подхода являются: стабильность результатов (приемлемые результаты получаются при любом качестве картинки) и гораздо более высокая скорость работы.

Создание маски на основе контура (алгоритм 1):

Шаг 0. Заполнить изображение-маску белым цветом.

Шаг 1. Положить `cur_line = 0`.

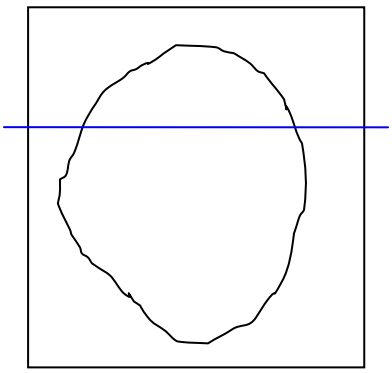
Шаг 2. В строке `cur_line` изображения найти первое (по индексу пикселя) сочетание пикселей «не зеленый – зеленый» (запомнить индекс – ind1) и последнее сочетание пикселей «зеленый – не зеленый» (запомнить индекс – ind2).

Шаг 3. В изображении маске в строке `cur_line` пиксели с номерами от `ind1` до `ind2` заполняем черным цветом.

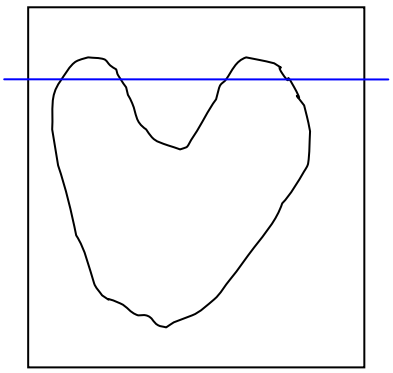
Если `cur_line = img.Width`, то конец.

Перейти к следующей строке изображения: `cur_line=cur_line+1` и перейти на Шаг 2.

Данный алгоритм предполагает «горизонтальную выпуклость» обрабатываемого контура, в том смысле, что горизонтальная линия, проведенная через все изображение, пересекает контур не более чем в двух точках (см. рисунок).



Корректный контур в смысле алгоритма 1



Некорректный контур в смысле алгоритма 1

Использование данного алгоритма ничем не ограничено в программе, т.к. функция шага 1 возвращает только «корректные» в смысле данного алгоритма контуры.

Возможности оптимизации алгоритма:

Реализация поиска зеленых пикселей с применением цепочечных инструкций процессора. Поиск основан на наличии в трехбайтовом представлении зеленого цвета байта со значением 0xFF. Алгоритм следующий:

1. Загрузка регистров указателями на изображения:

```
mov      eax,0xFF          ; ищем 0xFF
lea      edi,string        ; указатель на строку
mov      ecx,img.width     ; длина
go:
rep      scasb
jecxz    @1                ; достигли конца и ничего не нашли
```

2. Сравнение с маской 00 FF 00

```
;этот код выполняется, если 0xFF найдено
;здесь проверяем предыдущий и следующий за ним
movzx    eax,byte ptr [edi-2]
; байты (они д.б. Нулями т.к. Зеленый это 00,FF,00)
or       eax
; [edi-2] и [edi], тк цеп команды сдвигают
jnz      not_green
edi
; на одну позицию вперед от найденного номера
movzx    eax,byte ptr[edi]
or       eax
jnz      not_green
```

3. Проверка корректности номера

```
; Итак маска 00 FF 00 присутствует,
; определим «корректен» ли номер первого байта,
; который представляется edi-2, он должен делится на 3
mov      eax, edi-2
div      3
or       edx,edx
jz       got_it
not_green:                    ;здесь мы потерпели неудачу
;обработка
got_it:                            ;зеленый найден
```

Создание маски на основе контура (алгоритм 2):

Данный алгоритм использует для заполнения маски рекурсию.

Обозначения: *img[i][j]* – пиксель исходного изображения с координатами *(i,j)*, *msk[i][j]* - маски с координатами *(i,j)*.

Шаг 1. Положить x = img.Width/2, y = img.Height/2.

Шаг 2. Если img[x][y] не зеленый, то установить msk[x][y] черным,
иначе **конец** данной ветви рекурсии.

Шаг 3. Порождение ветвей рекурсии:
x=x-1, y=y-1, перейти к шагу 2,
x=x-1, y=y+1, перейти к шагу 2,
x=x+1, y=y+1, перейти к шагу 2,
x=x+1, y=y-1, перейти к шагу 2.

Алгоритм 2 работает с произвольными контурами, накладывая на контур только требование замкнутости. Функция шага 1 возвращает только замкнутые контуры, что позволяет использовать как алгоритм 1, так и алгоритм 2. Недостатками алгоритма 2 являются более низкая по сравнению с алгоритмом 1 скорость работы, из-за использования множества рекурсивных вызовов и потенциальная возможность переполнения при обработке больших изображений, тем не менее, этот алгоритм также реализован в программе.

Создание маски на основе вписанного эллипса:

В данном алгоритме в качестве приближенного контура лица берется эллипс, содержащийся в массиве с локализованным лицом. Данный метод позволяет получить приемлемый результат независимо от качества изображения. Заполнение контура черным цветом реализуется путем построения нескольких вписанных друг в друга эллипсов. Для построения эллипса используется алгоритм Брезенхема. Размеры эллипса вычисляются исходя из размеров прямоугольника, эксцентриситет эллипса рассчитывается на основе соотношений длин сторон прямоугольника. Коэффициенты пропорциональности были подобраны экспериментально.

Для создания целевой маски достаточно только скопировать часть исходного массива с лицом, соответствующую черной части черно-белой маски. Это с успехом выполняет функция ippiCopy_8u_C3RM, которая выполняет копирование на основе черно-белой маски.

4.7. Методы улучшения визуализации:

Для того чтобы граница наложения была не так заметна, в программе применяется три подхода:

- Зашумление контура

- Цветовая аппроксимация точек контура
- Применение эффектов наложения

Зашумление контура представляет собой некоторое прореживание точек границы, искусственное создание размытия границы на этапе создание черно-белой маски. Применяя такой прием можно улучшить отображаемый результат.

Цветовая аппроксимация дает гораздо более существенное улучшение качества отображения по сравнению с остальными методами, но требует больше вычислений. Ограничением этого метода является его применимость только в случае использования контуров на основе эллипсов. Суть метода состоит в обработке уже готового изображения, после применения наложения. В рамках готового кадра вычисляются точки эллипса-контура (который уже использовался при построении маски) и пиксели из некоторой окрестности этого контура аппроксимируются цветовыми значениями своих соседей «изнутри» и «снаружи».

На шаге наложения тоже можно предпринять попытки улучшения визуализации. Достигнуть этого можно, используя вместо черно-белой маски градиентную маску в оттенках серого, что тоже реализовано в программе. При использовании такой маски придется заменить вызов функции IPP для копирования в создании целевой маски и написать свой вариант этой процедуры, учитывая особенности маски.

5. Разработка пользовательского интерфейса и работа с видеопотоком.

Интерфейс пользователя демо-приложения FaceDisplacement был разработан с использованием библиотеки MFC 7.0. Большая часть исходного кода, реализующего интерфейс, была автоматически сгенерирована средствами среды разработки (Microsoft Visual C++ 2005).

Приложение построено на базе модального диалога, определяемого классом CFaceDisplacementDlg (наследуется от абстрактного класса CDialog). Этот класс описан в файле FaceDisplacementDlg.h, реализация его методов помещена в файл FaceDisplacementDlg.cpp.

При запуске приложения вызывается метод CFaceDisplacamentDlg::OnInit(), в котором производится инициализация приложения:

1. Инициализация элементов управления и полей класса их начальными значениями.
2. Создание статического буфера памяти для нужд библиотеки OpenCV (функция cvCreateMemStorage()).
3. Инициализация каскада классификаторов из файла “Cascade.xml” (функция cvHaarClassifierCascade()).
4. Подключение к Web-камере (функция cvCaptureFromCAM()), при этом создается объект типа CvCapture.
5. Создание окна вывода OpenCV с заголовком “Picture 1”, в которое будет производиться вывод готовых кадров.

6. Создание таймера с интервалом в 1 секунду (функция MFC CWnd::SetTimer()), по которому будет происходить обработка очередного кадра.

Соответственно, при выходе из приложения выполняются обратные действия: уничтожение таймера (функция MFC CWnd::KillTimer()), закрытие окна вывода OpenCV и отключение от Web-камеры с одновременным удалением объекта типа CvCapture (функция cvReleaseCapture()). Эти операции выполняются в методе OnCancel() класса CFaceDisplacementDlg.

Кнопки в 1-й части окна служат для выбора алгоритма распознавания лиц. При нажатии на соответствующую кнопку активизируется либо алгоритм, основанный на многоугольных контурах, либо алгоритм, основанный на эллипсах (вызывает функцию cvEllipse()).

2-я часть окна содержит элементы управления, предназначенные для корректирования специфических параметров отдельных алгоритмов. В этой версии программы можно изменять параметр Epsilon, определяющий степень погрешности контурного алгоритма распознавания. Значение можно изменять в интервале от 0.2 до 0.5 с шагом 0.05, текущая величина отображается над регулятором (который реализован посредством класса CSliderCtrl). При выборе алгоритма, базирующегося на эллипсах, бегунок становится неактивным, при выборе контурного алгоритма – снова активизируется. Переключение выполняется функцией MFC CWnd::EnableWindow().

Кнопки в 3-м блоке окна приложения служат для выбора одной из реализаций функций, выполняющих работу с изображениями. При нажатии на кнопку с надписью “IPP/MKL” производится поиск на компьютере установленных библиотек IPP и/или MKL компании Intel и (в случае положительного результата) использование этих функций из этих библиотек. При нажатии же на другую кнопку (с надписью “Internal”) обработка изображений выполняется с помощью внутренней реализации соответствующих функций в OpenCV.

При отсутствии на компьютере библиотек IPP/MKL оба режима работы эквивалентны с точки зрения производительности. В коде программы переключение реализаций происходит посредством вызова функции cvUseOptimized() с различным значением аргумента в обработчиках событий CFaceDisplacementDlg::OnBnClickedButtonIppMkl() и CFaceDisplacementDlg::OnBnClickedButtonInternal().

4-й блок окна управления является информационным. В нем показывается выбранный алгоритм и реализация функций. Под ними находится поле ввода, отображающее время обработки последнего кадра изображения в секундах. Все поля вводав приложении (служащие индикаторами) являются элементами управления класса CEdit с установленным флагом ReadOnly.

Основная работа происходит в обработчике события OnTimer() класса CFaceDisplacementDlg. При поступлении нового сообщения WM_Timer определяется выбранное с помощью бегунка значение параметра Epsilon и вывод его в соответствующее поле ввода.

Затем приложение выполняет захват с Web-камеры нового кадра путем последовательного вызова функций cvGrabFrame и cvRetrieveFrame. После этого вызывается функция GetTickCount() из Windows API, которая возвращает количество миллисекунд, прошедшее с момента запуска системы. Оно запоминается в переменной time_start типа DWORD.

После этого происходит обращение к пользовательской функции detect_and_draw(), которая обводит контуры лиц на рисунке и возвращает количество распознанных лиц. Если оно не менее 2, то вызывается пользовательская функция mySwitchFaces(), которая выполняет все действия по перестановке лиц. По завершению ее работы снова вызывается функция GetTickCount() и подсчитывается время, потраченное на обработку, путем вычитания значения time_start из возвращаемого результата.

Следующее действие – вызов функции myShowPicture, которая выводит обработанное изображение в окно OpenCV. Имя этого окна передается в функцию в качестве одного из ее параметров. Если же распознано менее 2 лиц, то новое изображение не формируется, а в окне OpenCV продолжает отображаться предыдущий кадр.

Наконец, происходит освобождение памяти, выделенной под временные динамические переменные, перевод времени из миллисекунд в секунды и вывод полученного значения в элемент CEdit в главном окне.