*Нижегородский государственный университет им. Н.И. Лобачевского*

Факультет вычислительной математики и кибернетики

# Учебный курс «Разработка мультимедийных приложений с использованием библиотек OpenCV и IPP»

# Лабораторная работа Сборка и установка библиотеки OpenCV. Использование библиотеки в среде Microsoft Visual Studio

*Кустикова В.Д.*

*При поддержке компании Intel*

Нижний Новгород

2012

# Содержание

# Введение

> Все видимое может быть также неувиденным, однако остается все же обнаруживаемым.
>
> *Эдмунд Гуссерль*

В настоящее время при решении многих практических задач используются системы компьютерного зрения (системы видеонаблюдения, управления процессами, организации информации и др.). OpenCV (Open Source Computer Vision Library) [1, 2] является одной из наиболее популярных библиотек компьютерного зрения с открытыми исходными кодами, в состав которой входит большое количество функций обработки изображений и видео в реальном времени. Библиотека реализована на языках C/C++, имеются обертки для вызова функций из языка Python, также существует кроссплатформенные .NET-обертки в составе EmguCV [3], позволяющие работать с OpenCV из C#, VB, VC++, IronPython и других языков платформы .NET. Нельзя не отметить, что в настоящее время активно развивается Java-интерфейс в связи с портированием библиотеки на мобильные платформы. Разработчики обеспечивают стабильную работу на базе операционных систем семейства Windows, Linux, MacOS, Android и iOS. Характерной особенностью OpenCV является модульность архитектуры, которая предполагает наличие нескольких статических или динамических библиотек. На данный момент доступно около двух десятков модулей, в частности, наиболее используемыми являются:

1. **`core`** – модуль, содержащий объявление всех структур данных, включая базовую структуру для представления многомерного массива **`Mat`** и функции работы с ней.
2. **`imgproc`** – модуль обработки изображений, который включает в себя линейную и нелинейную фильтрацию, геометрические преобразования изображений (в частности, масштабирование), преобразования цветовых пространств и т.д.
3. **`highgui`** – модуль, позволяющий отображать рабочие изображения, проигрывать видео и создавать простые интерфейсы управления.
4. **`ml`** – модуль, содержащий реализацию некоторых алгоритмов машинного обучения.
5. **`objdetect`** – модуль детектирования объектов (в частности, содержит реализацию детектора Виолы-Джонса или классификатора Хаара [18] и алгоритма детектирования объектов 20 разных классов (машина, человек, автобус и др.) Latent SVM [19]).

6. **`gpu`** – gpu-реализации некоторых алгоритмов, которые реализованы на центральном процессоре в других модулях.
7. **`video`** – модуль анализа видео, включающий функции оценивания движения на видео, вычитания фона и слежения за объектами на последовательности кадров видеопотока.
8. **`features2d`** – модуль выделения и сопоставления особых точек на изображениях.

В настоящей работе предполагается продемонстрировать использование некоторых базовых функций. Также подробно описать этап предварительной подготовки программной инфраструктуры с целью последующего использования библиотеки OpenCV на базе операционной системы Windows в среде Microsoft Visual Studio.

# 1. Методические указания

## 1.1. Цели и задачи работы

*Цель данной работы – рассмотреть технические этапы подготовки инфраструктуры и продемонстрировать использование базовых функций библиотеки OpenCV на простых практических примерах.*

Данная цель предполагает решение следующих задач:

1. Сборка и установка библиотеки OpenCV [1, 2] с использованием инсталлятора и из исходных кодов.
2. Настройка среды Microsoft Visual Studio с целью использования библиотеки при разработке C/C++ приложений.
3. Разработка приложения для определения контуров объектов, демонстрирующего применение некоторых базовых операций обработки изображений [20, 22]: загрузка и сохранение изображений, конвертирование цветового пространства, бинаризация изображения, выделение контуров объекта.
4. Разработка приложения для демонстрации базовых операций обработки видео на примере задачи детектирования лиц с помощью классификатора Хаара [18].

## 1.2. Структура работы

В работе предлагается описание возможных способов сборки и установки библиотеки OpenCV. Приводится последовательность действий, которые необходимо выполнить для настройки среды Microsoft Visual Studio при

разработке приложений с использованием функций библиотеки. Далее рассматриваются некоторые элементарные операции обработки изображений, решается задача выделения контуров объекта и разрабатывается приложение с целью освоения этих элементарных операций. Описываются некоторые операции работы с видеоданными, рассматривается задача видеодетектирования лиц с использованием классификатора Хаара. Разрабатывается приложение, которое демонстрирует применение некоторых функций работы с видео, а также реализации указанного детектора, входящей в состав библиотеки OpenCV.

## 1.3. Тестовая инфраструктура

Вычислительные эксперименты проводились с использованием следующей инфраструктуры (табл. 1).

Таблица 1. Тестовая инфраструктура

| | |
|---|---|
| Операционная система | Microsoft Windows 7 |
| Среда разработки | Microsoft Visual Studio 2010 |
| Библиотека TBB | Intel® Threading Building Blocks 3.0 for Windows, Update 3 (в составе Intel® Parallel Studio XE 2011 SP1) |
| Библиотеки OpenCV | Версия 2.4.2 |

## 1.4. Рекомендации по проведению занятий

При выполнении данной лабораторной работы рекомендуется следующая последовательность действий:

1. Напомнить вводную информацию о том, что такое компьютерное зрение, и какие основные задачи решаются средствами компьютерного зрения.
2. Выполнить обзор основных модулей библиотеки OpenCV, рассмотреть назначение и возможности каждого модуля.
3. Рассмотреть и продемонстрировать основные способы установки библиотеки OpenCV на системы с ОС семейства Windows.
4. Продемонстрировать последовательность действий, которые необходимо выполнить для создания и настройки консольного C/C++ проекта в среде Microsoft Visual Studio, чтобы подключить возможность использования библиотеки OpenCV.
5. Рассмотреть постановку и алгоритм решения задачи выделения контуров объекта на статическом изображении.

6. Рассмотреть базовые функции обработки изображений библиотеки OpenCV и разработать приложение для выделения контуров объекта на статическом изображении.
7. Рассмотреть постановку задачи детектирования лиц на видео, привести схему решения данной задачи посредством классификатора Хаара.
8. Объяснить базовые функции работы с видеопотоком, входящие в состав OpenCV, и разработать приложение, которое позволяет детектировать лица на видео с использованием классификатора Хаара, реализованного в OpenCV. Предполагается, что видео может быть получено из файла или с веб-камеры.

# 2. Сборка и установка библиотеки OpenCV под Windows

## 2.1. Основные способы установки

Выделяется два основных способа установки библиотеки OpenCV под Windows: посредством установочного файла и из исходных кодов. Первый является достаточно простым, интуитивно понятным и не требует значительных усилий. По существу предполагает распаковку архива, содержащего заголовочные файлы, исходные коды и бинарные файлы. Второй способ более предпочтителен, если интересно разобраться в том, как устроена библиотека в целом, чтобы перенять опыт распределенной разработки. При установке библиотеки из исходных кодов используется широко известная открытая утилита CMake [4]. Далее рассмотрим оба способа установки библиотеки и остановимся на некоторых особенностях каждого из них.

## 2.2. Установка OpenCV с использованием установочного файла

Чтобы установить библиотеку OpenCV посредством установочного файла, необходимо выполнить следующую последовательность действий:

1. Загрузить инсталлятор с официальной страницы проекта [2].
2. Запустить установочный файл. В результате появится окно, показанное на рис. 1.

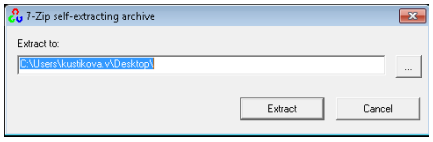


**Рис. 1.** Окно инсталлятора

3. Выбрать путь для извлечения файлов библиотеки и нажать кнопку «**Extract**», чтобы инициировать процесс извлечения (рис. 2). По окончании распаковки указанная на предыдущем шаге директория будет содержать необходимые заголовочные файлы (директория **build/include/opencv2**) и бинарные файлы библиотеки (**build**), а также исходные коды библиотеки (**modules**), набор примеров использования библиотечных функций (**samples**) и ряд других вложенных директорий, которые будут рассмотрены в работе по мере необходимости.

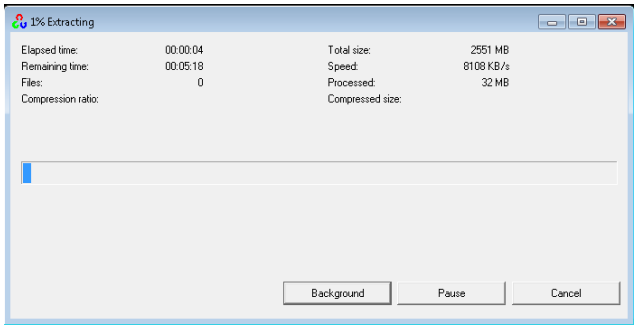


**Рис. 2.** Окно инсталлятора

Остановимся более подробно на составе директории **build**. Она содержит несколько вложенных директорий, в частности, **x86** и **x64**, в которых находятся сборки библиотеки для соответствующих архитектур. В каждой директории (**build/x86** и **build/x64**) имеются бинарные файлы, собранные под Microsoft Visual Studio 2008 (вложенная директория **vc9**), 2010 (**vc10**). Файлы с расширением **.dll** расположены в папке **bin**. Файлы с расширением **.lib** находятся в директории с одноименным названием. Заметим, что бинарные файлы, имеющие приставку **'d'** в конце названия, соответствуют **Debug**-сборке, остальные – сборке в режиме **Release**. Наряду с этим названия lib-файлов содержат набор цифр, которые определяют версию сборки библиотеки.

## 2.3. Сборка библиотеки OpenCV из исходных кодов

### 2.3.1. Утилита CMake

Утилита CMake [4] – это кроссплатформенная открытая система сборки. CMake применяется с целью управления процессом компиляции посредством использования конфигурационных файлов, не зависимых от компилятора. Утилита позволяет генерировать make-файлы и рабочие пространства среды, компилятор которой будет использован для сборки проекта, например, решение и набор проектов для Microsoft Visual Studio.

Схема работы утилиты CMake достаточно простая:

1. Разработчик описывает в специальном файле CMakeLists.txt параметры сборки (расположение исходных кодов, внешних библиотеки и т.д.).
2. CMake обрабатывает указанный файл и выдает файл с инструкциями сборки для конкретной платформы (например, make-файл GNU make или файл проекта Visual Studio). Далее, используя файл инструкций можно собирать проект.

Суть использования CMake состоит в том, что описание параметров сборки абстрагировано от платформы и особенностей расположения библиотек конкретных систем. Утилита имеет модульную архитектуру, за счет чего достигается кроссплатформенность. Условно архитектуру можно представить в виде нескольких блоков:

1. Модуль управления процессом сборки.
2. Модуль с реализациями базовых команд (поиск библиотеки в системе, добавление библиотеки в сборку и др.).
3. Модуль разбора файла, содержащего параметры сборки исходных кодов. Включает в себя классы, которые предназначены для хранения списка подключаемых библиотек и заголовочных файлов, также другие опции.
4. Модуль генерации платформо-зависимых файлов сборки исходных кодов, которые впоследствии могут быть использованы для компиляции и линковки исходных кодов системы.

Отметим, что такая архитектура утилиты позволяет расширять функционал, как следствие, реализовывать новые генераторы файлов сборки исходных кодов.

По существу CMake позволяет разработчикам программных систем решать несколько основных проблем:

1. Разрешение зависимостей между отдельными частями проекта.
2. Синтаксическая сложность разработки make-файлов, т.к. CMake снимает с разработчика необходимость написания make-файлов.
3. Обеспечение переносимости системы на другие платформы, что достигается за счет платформенной независимости файла параметров сборки.

### 2.3.2. Установка CMake

Для установки утилиты CMake необходимо выполнить следующие действия:

1. Загрузить установочный файл с официальной страницы проекта [4].
2. Запустить инсталлятор. При запуске появится окно, показанное на рис. 3.

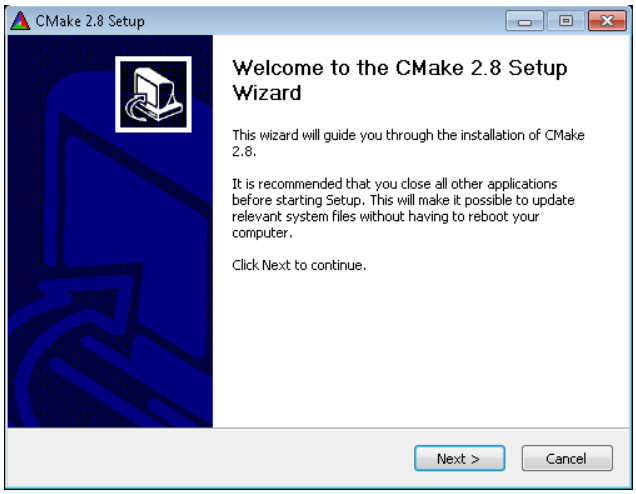


**Рис. 3.** Окно инсталлятора

3. Нажать кнопку «**Next**», чтобы перейти на следующий этап процедуры инсталляции. В результате можно видеть окно, показанное на рис. 4, которое содержит лицензионное соглашение.

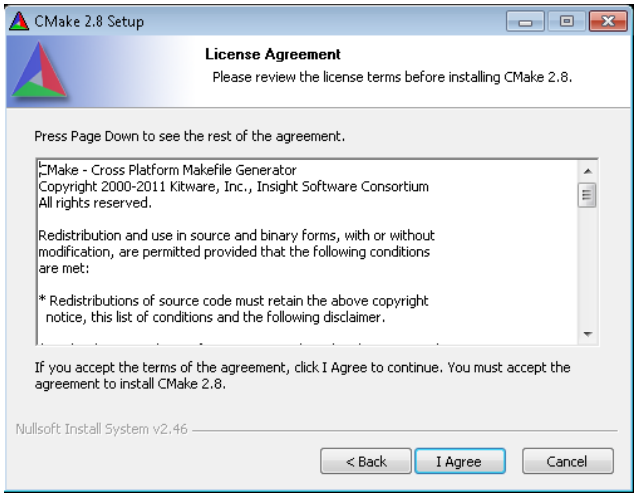


**Рис. 4.** Окно инсталлятора

4. Нажать кнопку «**I agree**», чтобы согласиться с условиями лицензионного соглашения. Далее появится окно, показанное на рис. 5, в котором предлагается прописать путь до исполняемого файла утилиты CMake в переменную окружения PATH и создать ярлык на рабочем столе для быстрого запуска указанной утилиты.

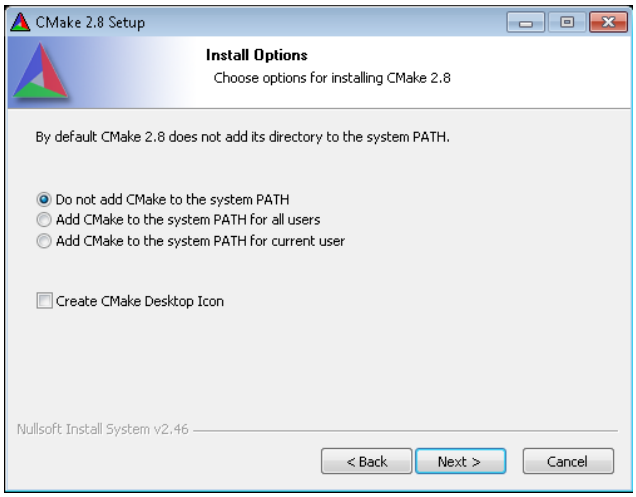


**Рис. 5.** Окно инсталлятора

5. Выбрать необходимые опции и нажать кнопку «**Next**», в результате чего пользователю будет показано окно, приведенное на рис. 6.

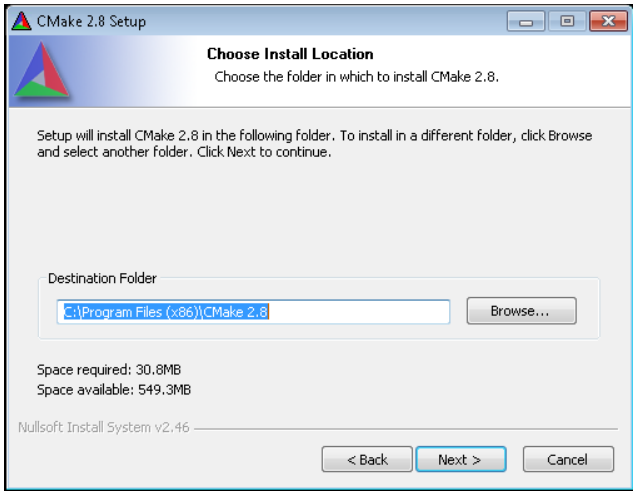


**Рис. 6.** Окно инсталлятора

6. Выбрать директорию для установки утилиты и нажать кнопку «**Next**». Далее появится окно (рис. 7), в котором предлагается указать название директории, которая будет содержать ярлыки утилиты в меню «**Пуск**».

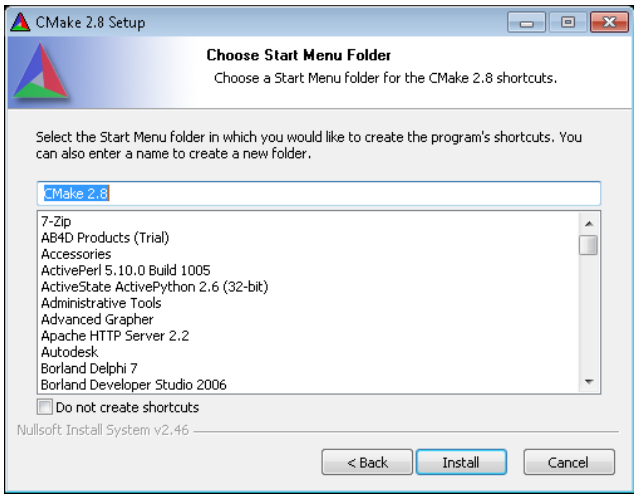


**Рис. 7.** Окно инсталлятора

7. Указать название директории и нажать кнопку «**Install**». Данное действие инициирует процесс установки, в результате которого появится окно завершения процесса (рис. 8).

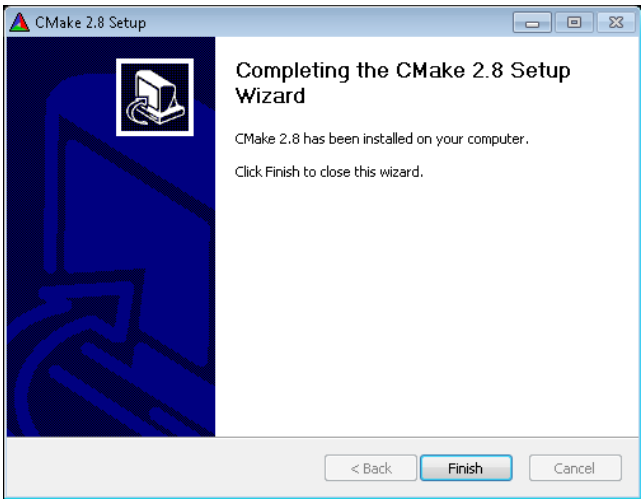


**Рис. 8.** Окно инсталлятора

8. Нажать кнопку «**Finish**» для завершения установки утилиты.

### 2.3.3. Сборка OpenCV с использованием CMake

В данном разделе предлагается описание последовательности действий, выполнение которых необходимо для сборки библиотеки OpenCV с использованием утилиты CMake согласно руководству [8]. Несмотря на то, что CMake имеет интуитивно понятный графический интерфейс под Windows, далее описывается последовательность команд командной

строки, т.к. их достаточно просто можно сопоставить с командами других платформ, как следствие, выполнить сборку OpenCV по той же схеме.

1. Создать директорию, которая будет содержать исходные коды библиотеки, например, с названием **OpenCV-2.4.2** в соответствии с загружаемой версией.

```
c:\temp>mkdir OpenCV-2.4.2
```

2. Загрузить последнюю версию исходных кодов библиотеки с сервера проекта [5, 6]. Отметим, что для этого предварительно необходимо установить систему управления версиями Subversion [7] или любой другой клиент [1] в зависимости от того, какая система используется разработчиками библиотеки на момент загрузки. Ниже приведена командная строка, позволяющая загрузить исходные коды с использованием Subversion.

```
c:\temp\OpenCV-2.4.2>"C:\Program Files (x86)\Subversion\bin\svn.exe" co http://c
ode.opencv.org/svn/opencv/trunk/opencv
A    opencv\android
A    opencv\android\refman.rst
A    opencv\android\service
```

   В результате загрузки в **OpenCV-2.4.2** будет создана вложенная директория **opencv**, в состав которой входят исходные коды и скрипты, необходимые для сборки решения.

   Остановимся на расположении заголовочных файлов. Файл **opencv.hpp**, в котором подключаются заголовочные файлы всех модулей библиотеки, расположен в директории **opencv/include/opencv2**. Заголовочные файлы модулей находятся по путям **opencv/modules/<имя модуля>/include**.

3. Опционально установить дополнительные программное обеспечение и библиотеки, которые могут потребоваться в процессе сборки OpenCV:

   - Python версии 2.6.x или 2.7.x и NumPy, если необходимо собрать обертки для Python;
   - Intel Threading Building Blocks (TBB) – библиотека разработки параллельных приложений, средства которой используются разработчиками OpenCV для распараллеливания алгоритмов на системы с общей памятью;
   - Qt версии 4.6 обеспечивает функции для создания кроссплатформенного (Windows, Linux, Mac) графического интерфейса;
   - Intel Parallel Primitives (IPP) с версии 5.1 до 6.1 для ускорения некоторых вычислительных операций (конвертирование цвета,

[1] Subversion – свободная централизованная система управления версиями. Аналогами являются такие системы, как Git, Mercurial, Bazaar, Tortoise SVN (имеет графический интерфейс).

тренировка классификатора Хаара, функции вычисления быстрого преобразования Фурье (БПФ));
- LiveTeX (или MiKTeX под Windows, MacTeX под Mac) и Sphinx, чтобы собрать документацию OpenCV в форматах PDF и HTML;
- последняя версия NVIDIA CUDA Toolkit, если требуется собрать GPU модуль библиотеки;
- другое ПО и библиотеки, специфичные для Unix-систем, MacOS, Android и iOS.

4. Создать директорию, например, **bin**, в которую утилита CMake будет генерировать файлы сборки исходных кодов.

```
c:\temp\OpenCV-2.4.2>mkdir bin
```

5. Установить созданную директорию в качестве рабочей.

```
c:\temp\OpenCV-2.4.2>cd bin
c:\temp\OpenCV-2.4.2\bin>_
```

6. Запустить исполняемый файл **cmake** утилиты CMake[2] и создать файлы описания сборки.

   Общий вид командной строки генерации приведен ниже. Также указаны две наиболее важные опции **cmake**, которые будут использованы далее. Полный список опций можно обнаружить, если выполнить команду **cmake --help**.

```
cmake [options] <source_dir>

[options] – параметры конфигурации CMake и сборки OpenCV
<source_dir> - директория с исходными кодами библиотеки

Наиболее важные опции:
-D <var>:<type>=<value> используется для установки значений переменных из файла описания сборки исходных кодов
-G <generator-name> используется для указания имени генератора (полный список генераторов, доступных для данной платформы, можно найти в справке к cmake)
```

   Сгенерируем решение для компиляции OpenCV в Visual Studio 2010 с поддержкой параллелизма средствами библиотеки TBB, дополнительно подключим использование IPP, также соберем примеры использования функций. Для этого необходимо выполнить команду, показанную ниже.

[2] Директория утилиты CMake среди бинарных файлов содержит файл cmake-gui, который позволяет использовать CMake в графическом режиме.


Отметим, что если при инсталляции утилиты CMake были выбраны опции установки по умолчанию и путь до исполняемых модулей утилиты не был прописан в переменную окружения **PATH**, то при вызове необходимо указать полный путь до **cmake.exe** ровно так, как показано выше.

Разберем подробнее назначение каждой опции командной строки. Первый параметр **BUILD_EXAMPLES** означает, что в решение необходимо включить проекты с примерами использования различных модулей OpenCV.

```
-D BUILD_EXAMPLES=ON
```

Опция **WITH_TBB=YES** включает сборку библиотеки с TBB.

```
-D WITH_TBB=YES
```

Далее следует набор параметров, связанных с расположением файлов библиотеки TBB: **TBB_INCLUDE_DIRS** – полный путь до заголовочных файлов библиотеки, **TBB_LIB_DIR** – полный путь до lib-файлов TBB, **TBB_STDDEF_PATH** – полный путь до файла **tbb_stddef.h** с объявлениями констант и некоторых типов.

```
-D TBB_INCLUDE_DIRS="c:\Program Files (x86)\Intel\Composer
XE 2011 SP1\tbb\include"
-D TBB_LIB_DIR="c:\Program Files (x86)\Intel\Composer XE
2011 SP1\tbb\lib\ia32\vc9"
-D TBB_STDDEF_PATH="c:\Program Files (x86)\Intel\Composer
XE 2011 SP1\tbb\include\tbb\tbb_stddef.h"
```

Следующие два параметра аналогично позволяют подключить использование IPP. Отметим, что все пути, которые указываются в опциях **cmake**, не должны содержать в конце обратный слеш, в этом случае парсер утилиты не обрабатывает эти опции.

```
-D WITH_IPP=YES
-D IPP_H_PATH="C:/Program Files (x86)/Intel/Composer XE
2011 SP1/ipp/include"
```

Опция **-G** означает, что далее будет указано имя генератора файла сборки. В данном случае используется генератор Visual Studio 10, т.е. будет сформировано решение Visual Studio 2010 для компиляции под 32-битную архитектуру.

```
-G "Visual Studio 10"
```

Обратите внимание на стандартный поток вывода, который формируется в результате исполнения указанной выше команды. В особенности на блок, в котором приведен список сторонних библиотек, использованных при генерации файла сборки. Примерный вид того, что должно получиться, приведен ниже.

```
--   Other third-party libraries:
--     Use IPP:                     7.0 [7.0.205]
--          at:                     C:/Program Files (x86)/Intel/Composer XE 201
1 SP1/ipp
--     Use TBB:                     YES (ver 4.0 interface 6004)
--     Use Cuda:                    NO
--     Use Eigen:                   NO
--     Use Clp:                     NO
```

После исполнения команды на формирование файлов описания сборки в директории **bin** можно обнаружить файл решения (с расширением **.sln**) и набор файлов проектов (**.vcxproj**), входящих в состав этого решения (рис. 9).

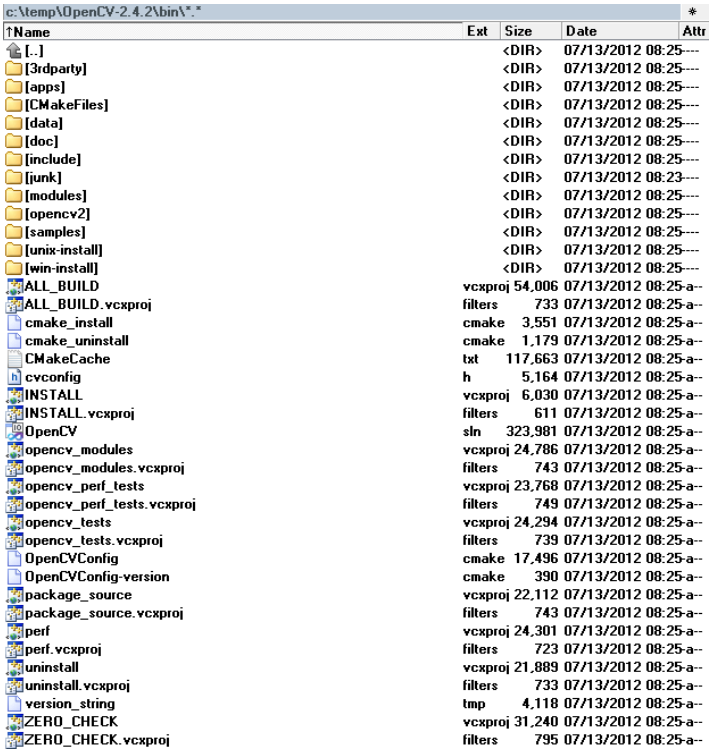


**Рис. 9.** Структура сгенерированных файлов для сборки OpenCV из Microsoft Visual Studio 2010

7. Открыть решение **OpenCV.sln**.
8. Выбрать режим компиляции (**Debug** или **Release**) в выпадающем списке на панели инструментов (рис. 10).


**Рис. 10.** Панель инструментов для смены режима компиляции

9. Выполнить команду **Rebuild** контекстного меню, которое выпадает при нажатии правой кнопки мыши на проекте **`ALL_BUILD`** для перекомпиляции всего решения.

В результате компиляции в директории, содержащей решение, будет создано две вложенных директории **bin** и **lib**, каждая из которых в зависимости от выбранного режима компиляции содержит папку **Debug** или **Release** с набором dll- и lib-файлов. Аналогично случаю установки с помощью инсталляционного файла, бинарные файлы, имеющие приставку **'d'** в конце названия, соответствуют **Debug**-сборке, остальные – сборке в режиме **Release**.

### 2.3.4. Структура проектов в решении Microsoft Visual Studio для OpenCV

На данном этапе перейдем к рассмотрению основных проектов в решении и непосредственной компиляции библиотеки OpenCV. Настроим отображение панели **Solution Explorer**, выполнив команду главного меню **View→Solution Explorer**.

В корне решения находится один единственный проект **`ALL_BUILD`** и множество папок с проектами, содержащими разного рода функционал. Проект **`ALL_BUILD`** установлен как рабочий и отвечает за компиляцию всех проектов в решении, как было замечено в предыдущем разделе. По умолчанию решение собирается в **Debug**-режиме. Если необходимо собрать решение в **Release**, то следует выбрать соответствующий режим в выпадающем списке на панели инструментов. Похожие действия (изменение конфигурации с **Win32** на **x64**) не сработают, если требуется собрать 64-битную версию библиотеки. В этом случае необходимо заново сгенерировать решение, но при исполнении команды **`cmake`** указать другое имя генератора (например, Visual Studio 10 Win64).

Рассмотрим подробнее назначение и содержимое некоторых папок в решении:

- **`3rdparty`** содержит набор проектов с исходными кодами сторонних библиотек (**`libjasper`** [9], **`libjpeg`** [10], **`libpng`** [11], **`libtiff`** [12] – кодеки для работы с изображениями в соответствующих форматах, **`zlib`** [13] – библиотека для сжатия данных);
- **`modules`** составлена из рабочих модулей библиотеки OpenCV;
- **`samples`** – папка, содержащая множество примеров использования разных функций библиотеки;

- **`tests accuracy`** содержит автоматические тесты для проверки корректности работы алгоритмов;
- **`tests performance`** – набор тестов производительности программных реализаций некоторых алгоритмов.

# 3. Подготовка среды Microsoft Visual Studio для разработки приложений с использованием OpenCV

## 3.1. Создание проекта

Прежде всего, создадим новое **Решение (Solution)**, в которое включим первый **Проект (Project)** данной лабораторной работы. Последовательно выполните следующие шаги:

1. Запустите приложение Microsoft Visual Studio 2010.
2. В меню **File** выполните команду **New→Project…**.
3. Как показано на рис. 11, в диалоговом окне **New Project** в типах проекта выберите **Win32**, в шаблонах **Win32 Console Application**, в поле **Name** введите название проекта (для каждого приложения будет использовано свое название), в поле **Solution Name** – название решения **01_OpenCV**, в поле **Location** укажите путь к папке с лабораторными работами. Нажмите OK.

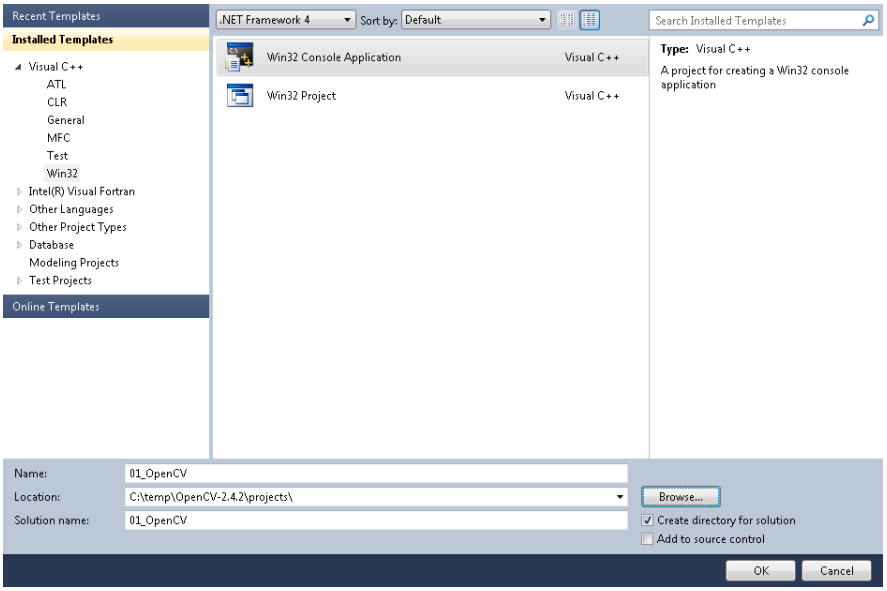


**Рис. 11.** Создание решения для лабораторной работы

4. В диалоговом окне **Win32 Application Wizard** нажмите **Next** (или выберите **Application Settings** в дереве слева) и установите флаг **Empty Project**. Нажмите **Finish**.
5. В окне **Solution Explorer** в папке **Source Files** выполните команду контекстного меню **Add→New Item…**. В дереве категорий слева выберите **Code**, в шаблонах справа – **C++ File (.cpp)**, в поле **Name** введите имя файла **main**. Нажмите **Add**. В результате выполненной последовательности действий в окне редактора кода Visual Studio будет открыт пустой файл **main.cpp**.

Далее создадим заготовку функции **`main()`** с параметрами командной строки. Как правило, в разрабатываемых приложениях в качестве параметров будет приниматься название изображения или видео, а также параметры тех или иных алгоритмов.

```
int main(int argc, char *argv[])
{
      // TODO: source code
      return 0;
}
```

## 3.2. Настройка свойств проекта

Процесс настройки свойств проекта сводится к выполнению трех действий:

1. Установка путей до заголовочных файлов библиотеки OpenCV. Выполните команду контекстного меню **Properties**, чтобы получить доступ к настройкам проекта. Откройте вкладку **Configuration Properties→C/C++→General** (рис. 12). Сверху в окне свойств в выпадающем списке **Configuration** выберите значение **All Configurations**, чтобы установить свойство для всех режимов компиляции (**Debug** и **Release**). В поле **Additional Include Directories** укажите пути до заголовочных файлов библиотеки OpenCV. В предыдущих разделах при описании процедур установки отдельно были отмечены директории, содержащие заголовочные файлы библиотеки OpenCV. Они различаются в зависимости от типа установки. Нажмите кнопку **Apply**, чтобы применить указанное свойство.

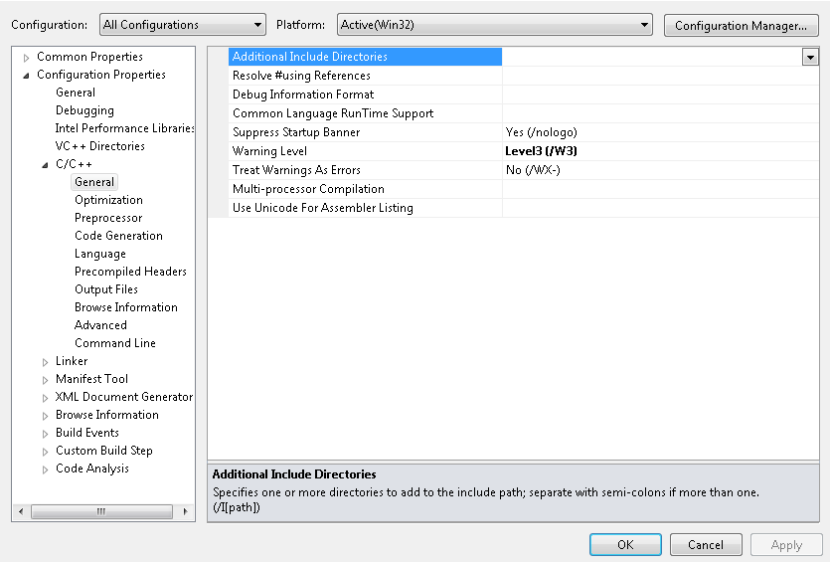


**Рис. 12.** Окно свойств проекта для установки путей до заголовочных файлов подключаемых библиотек

2. Установка путей до подключаемых библиотек (lib-файлов). Откройте вкладку **Configuration Properties→Linker→General** (рис. 13). Сверху в окне свойств в выпадающем списке **Configuration** выберите значение **Debug**. В поле **Additional Library Directories** укажите путь до lib-файлов, скомпилированных в режиме **Debug**. В предыдущих разделах при описании процедур установки отдельно были отмечены директории, содержащие lib-файлы библиотеки OpenCV. Они различаются в зависимости от типа установки. В окне свойств в выпадающем списке **Configuration** выберите значение **Release** и установите пути до lib-файлов, собранных в режиме **Release**.

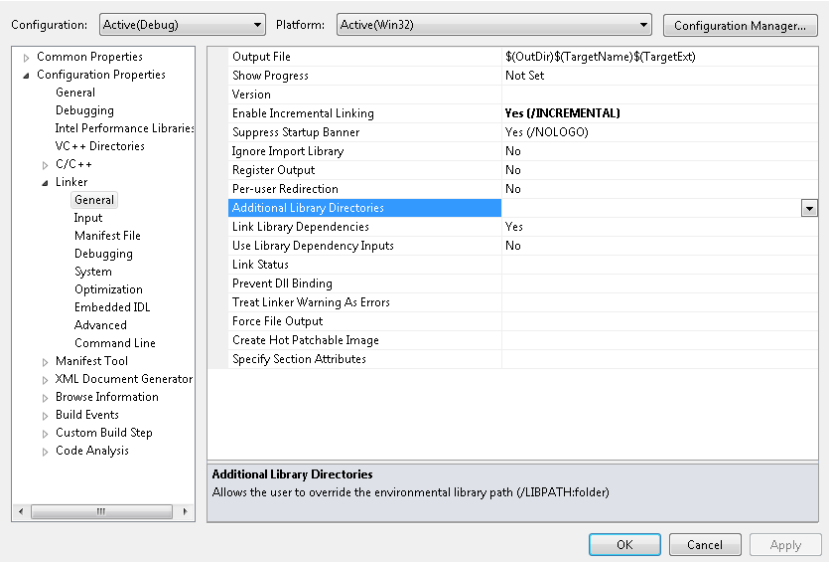


**Рис. 13.** Окно свойств проекта для установки путей до подключаемых библиотек

3. Указание списка подключаемых программных библиотек. Откройте вкладку **Configuration Properties→Linker→Input** (рис. 14). Сначала в окне свойств в выпадающем списке **Configuration** выберите значение **Debug** и установите в поле **Additional Dependencies** список lib-файлов. Схема именования lib-файлов следующая: **opencv_<мод><вер>d.lib**, где **<мод>** – название подключаемого модуля, **<вер>** – версия библиотеки (например, **opencv_core249d.lib**). Аналогичные действия необходимо проделать для режима **Release**, указав список lib-файлов с названиями вида **opencv_<мод><вер>.lib** (например, **opencv_core249.lib**). Отметим, что можно подключать не все модули библиотеки, а только те, функционал которых будет использован в процессе разработки приложений.

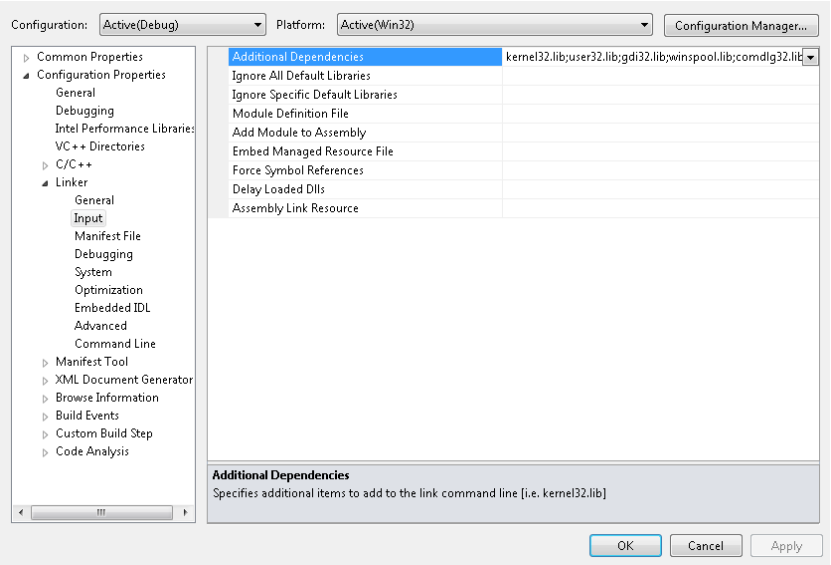


**Рис. 14.** Окно свойств проекта для установки списка подключаемых библиотек

## 3.3. Подключение заголовочных файлов в исходном коде приложения

Чтобы использовать функции библиотеки OpenCV при разработке собственных приложений, достаточно подключить заголовочный файл **`opencv.hpp`**, содержащий подключение большинства установленных модулей библиотеки, и пространство имен **`cv`**, в которое заключены все функции библиотеки.

```
#include <opencv2\opencv.hpp>
using namespace cv;
```

Если заранее известно, что в процессе разработки будет использован функционал конкретного набора модулей, то можно подключить только заголовочные файлы соответствующих модулей, например:

```
#include <opencv2\core.hpp>
#include <opencv2\objdetect.hpp>

using namespace cv;
```

## 3.4. Компиляция и запуск программы. Возможные проблемы и пути их решения

Чтобы протестировать корректность настроек, подключите заголовочные файлы библиотеки OpenCV в файл **main.cpp** в соответствии с указаниями

раздела 3.3 и скомпилируйте программу посредством нажатия клавиши **F7**, или используя пункт **Build** главного меню.

В процессе компиляции могут возникнуть следующие проблемы:

1. Ошибка открытия заголовочного файла. Примерный текст ошибки приведен ниже. Суть проблемы состоит в том, что указаны неправильные пути до подключаемых заголовочных файлов библиотеки. Решение – проверить в настройках проекта корректность путей до заголовочных файлов (см. указания раздела 3.2).

```
fatal error C1083: Cannot open include file:
'opencv2/opencv.hpp': No such file or directory
```

2. Ошибка линковки с модулем библиотеки OpenCV. Текст ошибки приведен ниже. Проблема состоит в том, что указаны неправильные пути до подключаемых lib-файлов. Решение – проверить в настройках проекта корректность путей до lib-файлов (см. раздел 3.2).

```
LINK : fatal error LNK1104:
cannot open file 'opencv_core242d.lib'
```

3. Ошибки линковки, связанные с отсутствием реализации некоторых функций библиотеки OpenCV. Пример такой ошибки приведен далее. Проблема состоит в том, что не подключен модуль (lib-файл), содержащий реализацию указанной в тексте ошибки функции. Решение – добавить в список lib-файлов необходимый модуль (см. раздел 3.2).

```
error LNK2019: unresolved external symbol
"void __cdecl cv::line(class cv::Mat &,
class cv::Point_<int>,class cv::Point_<int>,
class cv::Scalar_<double> const &,int,int,int)"
(?line@cv@@YAXAAVMat@1@V?$Point_@H@1@1ABV?$Scalar_@N@1@HHH@
Z) referenced in function _main
```

Когда программа скомпилировалась, запустите ее. Для этого можно нажать сочетание клавиш **Ctrl+F5**, либо воспользоваться пунктом **Start Without Debugging** главного меню. При попытке запуска может возникнуть ошибка, показанная на рисунке (рис. 15).

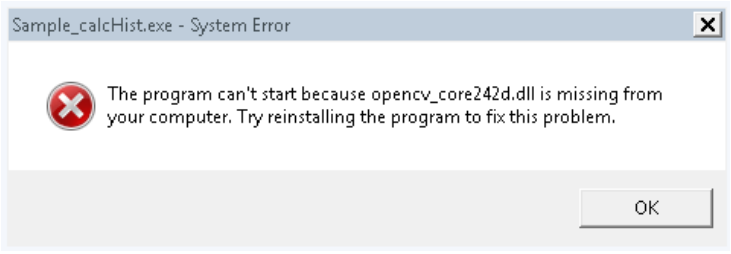


**Рис. 15.** Ошибка исполнения

Проблема в том, что при исполнении выполняется поиск dll-файлов библиотеки OpenCV, и среда не находит указанные бинарные файлы. Существует два варианта решения данной проблемы:

1. Задать в переменной окружения **PATH** пути до dll-файлов библиотеки OpenCV. В зависимости от способа установки библиотеки пути необходимо указать тот или иной путь (см. разделы 2.2 или 2.3).
2. Скопировать указанный dll-файл к исполняемым файлам программы. Очевидно, что такой dll-файл будет не единственный, поэтому требуется последовательно копировать запрашиваемые файлы. Заметим, что если библиотека OpenCV собрана с поддержкой параллелизма, то также потребуется скопировать dll-файлы библиотеки Intel Threading Building Blocks (**tbb_debug.dll**, **tbbmalloc_debug.dll** или **tbb.dll**, **tbbmalloc.dll**), при условии, что TBB не установлена на рабочей машине, либо переменная окружения **PATH** не содержит путь до dll-файлов, входящих в ее состав.

# 4. Разработка приложения для демонстрации базовых операций работы с изображениями

## 4.1. Задача определения контуров объектов

К настоящему моменту решены все технические вопросы, поэтому в данном разделе сконцентрируемся на модельной задаче определения контуров объекта, а также на рассмотрении базовых функций библиотеки OpenCV для решения данной задачи.

Резкое изменение яркости изображения представляет интерес по нескольким причинам [20]:

1. Такие изменения, как правило, возникают на границах объектов, например, при изображении светлого объекта на темном фоне или наоборот.
2. Резкие изменения яркости могут быть следствием изменения отражательной способности на достаточно характерных структурах, таких, как разметка пешеходного перехода или пятна на шкуре леопарда.
3. Резкие изменения ориентации поверхности – это еще одна возможная причина изменения яркости.

Пользуясь терминологией, принятой в компьютерном зрении, точки, в которых происходит резкий перепад яркости изображения, называются *краями* или *краевыми точками*. Возникает проблема, связанная с тем, как связать краевые точки с границами или контурами объектов на

изображении. Далее попробуем решить данную задачу средствами библиотеки OpenCV. Предварительно рассмотрим ряд базовых функций работы с изображениями.

## 4.2. Базовые операции

### 4.2.1. Создание изображения

В OpenCV любое изображение представляет собой матрицу интенсивностей, а именно объект класса **Mat**. Существует много способов создания объекта данного типа, рассмотрим два наиболее часто используемых в контексте изображений:

1. Конструктор класса **Mat**.

```
// _rows – количество строк
// _cols – количество столбцов
// _type – тип матрицы (CV_8UC1, CV_64FC3 и другие)
Mat(int  rows, int  cols, int  type);
Mat(Size _size, int _type);

// _s - заполнитель
Mat(int _rows, int _cols, int _type, const Scalar& _s);
Mat(Size _size, int _type, const Scalar& _s);
```

2. Метод **create**.

```
void create(int _rows, int _cols, int _type);
void create(Size _size, int _type);
```

По сути это два эквивалентных способа создания, выбор, что использовать, безусловно, принадлежит разработчику.

После выполнения всех операций с объектом типа **Mat** необходимо освободить память, вызвав метод **release**.

```
void release();
```

### 4.2.2. Загрузка изображения

Загрузка изображения – это одна из основных операций. Ниже приведен прототип функции OpenCV, которая отвечает за загрузку изображений.

```
Mat imread(const string& filename, int flags=1)
```

Как отмечалось в предыдущем разделе, изображение представляет собой двумерную матрицу интенсивностей, поэтому функция **imread** возвращает объект типа **Mat**. В качестве входных параметров функция принимает название файла **filename** и целочисленный флаг **flags**, определяющий правило загрузки:

- **flags>0** означает, что изображение загружается как цветное трехканальное. Отметим, что в OpenCV каналы хранятся в порядке BGR;
- **flags<0** – изображение загружается так, как есть, т.е. автоматически определяется количество каналов;
- **flags=0** используется для загрузки изображения в оттенках серого, что позволяет сразу получить из цветного изображение в оттенках серого без дополнительной конвертации.

Класс **Mat** имеет набор полей и методов, часть из которых представляет интерес с точки зрения работы с изображением:

- **uchar* data** – поле, содержащее значение интенсивностей для каждого пикселя изображения. Если данное поле после загрузки изображения имеет значение **NULL**, значит, произошла ошибка в процессе загрузки изображения. Типичными ошибками является неправильный путь или название изображения, неподдерживаемый или неправильный формат, некорректные права доступа к файлу;
- **int rows, cols** – количество строк и столбцов в матрице;
- **int channels() const** – метод, который возвращает количество каналов в изображении;
- **Size size() const** – метод для получения размера изображения (**width**, **height** – поля класса **Size**). Обратим внимание читателя, что **rows** и **cols** совпадают с **width**, **height** соответственно.

Описание назначения других полей и методов можно найти в документации [14].

### 4.2.3. Сохранение изображения

Естественным образом раз есть операция чтения изображения, необходима и противоположная операция сохранения. Сохранение изображения в файл осуществляется посредством вызова функции **imwrite**, прототип которой приведен далее.

```
bool imwrite(const string& filename,
             const Mat& img,
             const vector<int>& params=vector<int>())
```

Функция в качестве параметров принимает название файла **filename**, в который необходимо сохранить изображение, само изображение **img**, а также вектор целочисленных параметров **params**. Указанный вектор определяет параметры сохранения в файл, специфичные для выбранного формата сохранения. Формат определяется расширением файла **filename**. Вектор параметров представляет собой единую последовательность пар **<идентификатор_параметра>** и

**<значение_параметра>**. На данный момент доступны следующие идентификаторы:

- **CV_IMWRITE_JPEG_QUALITY** – качество сохранения изображения в формате JPEG, принимает значения от 0 до 100%, по умолчанию значение равно 95%.
- **CV_IMWRITE_PNG_COMPRESSION** – уровень сжатия изображения при сохранении в формате PNG, принимает значения от 0 до 9 (чем больше уровень, тем мельче результирующее изображение), по умолчанию используется уровень сжатия, равный 3.
- **CV_IMWRITE_PXM_BINARY** – флаг, который определяет тип хранения (бинарный или нет) изображения в форматах PPM, PGM, PBM, принимает значение 0 или 1, по умолчанию используется 1.

### 4.2.4. Отображение изображения

Безусловно, после обработки изображения необходимо посмотреть результат. Для корректного отображения изображения необходимо последовательно выполнить три действия:

1. Создать окно для отображения с помощью функции **namedWindow**. Функция принимает на вход название окна, которое служит идентификатором окна, и флаги. Отметим, что если окно с аналогичным идентификатором уже существует, то функция не выполняет никаких действий.

```
void namedWindow(const string& winname, int flags)
```

   Параметр **flags** может принимать следующие значения или их комбинации, полученные в результате применения оператора '**OR**':

   - **CV_WINDOW_NORMAL** или **CV_WINDOW_AUTOSIZE**, первое значение позволяет пользователю вручную изменять размеры окна в процессе работы приложения, в то время, как второе автоматически подгоняет размеры окна под размеры изображения, запрещая пользователю изменять размер вручную;
   - **CV_WINDOW_FREERATIO** или **CV_WINDOW_KEEPRATIO**, **CV_WINDOW_FREERATIO** подгоняет изображение под окно без сохранения пропорций, **CV_WINDOW_KEEPRATIO** позволяет сохранять пропорции;
   - **CV_GUI_NORMAL** или **CV_GUI_EXPANDED**, первое значение обеспечивает отрисовку окна без дополнительных компонент таких, например, как строка состояний и панель инструментов, второе же, напротив, предоставляет более новые возможности по созданию графических интерфейсов пользователя наряду с отображением изображений.

По умолчанию для окна устанавливаются свойства **CV_WINDOW_AUTOSIZE**, **CV_WINDOW_KEEPRATIO**, и **CV_GUI_EXPANDED**.

2. Отобразить изображение посредством вызова функции **imshow**. Для отображения необходим идентификатор окна **winname**, в которое будет помещено изображение, а также само изображение **image**.

```
void imshow(const string& winname, const Mat& image)
```

3. Дождаться нажатия какой-либо клавиши, чтобы закрыть окно. Ожидание реализуется посредством вызова функции **waitKey**, которая принимает на вход время ожидания **delay** в миллисекундах. По умолчанию **delay=0**, что означает ожидание в течение бесконечного промежутка времени, т.е. исполнение приложения будет продолжено после нажатия какой-либо клавиши. Отметим, что отсутствие вызова указанной функции приведет к тому, что по завершении программы окно будет автоматически закрыто.

```
int waitKey(int delay=0)
```

### 4.2.5. Копирование изображения

В некоторых случаях при разработке приложений необходимо работать не с самим изображением, а с его полной копией, чтобы повторно использовать исходное изображение. Напомним, что изображение представляется объектом класса **Mat**. В состав методов данного класса входит метод **clone**, который решает указанную задачу.

```
Mat clone() const;
```

Есть еще пара полезных методов копирования **copyTo**: первый (см. прототип ниже) выделяет память для хранения полностью аналогичной матрицы **m** и копирует в нее содержимое текущей матрицы, второй метод выполняет копирование в соответствии с передаваемой маской **mask** (копируются только элементы, которым соответствуют ненулевые элементы маски).

```
void copyTo(Mat& m) const;
void copyTo(Mat& m, const Mat& mask) const;
```

### 4.2.6. Конвертирование изображения в другое цветовое пространство

Конвертирование изображения из одного цветового пространства в другое – это еще одна достаточно важная операция, т.к., например, многие алгоритмы компьютерного зрения работают на изображениях в оттенках серого, в то время как исходное изображение с камеры цветное. Функция **cvtColor** позволяет конвертировать изображение в другое цветовое пространство.

```
void cvtColor(const Mat& src, Mat& dst,
              int code, int dstCn=0)
```

Входными параметрами данной функции являются исходное изображение **src**, которое необходимо конвертировать в другое цветовое пространство, конвертированное изображение **dst**, код операции конвертирования **code** (из какого в какое пространство) и количество каналов в результирующем изображении **dstCn**. По умолчанию параметр **dstCn** принимает значение 0, это означает, что количество каналов результирующего изображения будет определено автоматически. Отметим, что в случае, когда исходное изображение цветное, явно указывается порядок сохранения каналов (RGB или BGR). Рассмотрим подробнее возможные значения, которые принимает параметр **code** [15]:

- **CV_RGB2GRAY**, **CV_GRAY2RGB** – конвертирование из RGB-пространства в оттенки серого (значение интенсивности вычисляется как взвешенная линейная свертка интенсивностей по всем трем каналам) и наоборот (дублирование интенсивности по трем каналам);
- **CV_BGR2XYZ**, **CV_RGB2XYZ**, **CV_XYZ2BGR**, **CV_XYZ2RGB** – преобразование из RGB/BGR-пространств в линейное трехкомпонентное пространство CIE XYZ, основанное на результатах измерения характеристик человеческого глаза, и обратно;
- **CV_BGR2YCrCb**, **CV_RGB2YCrCb** – конвертирование изображения из BGR/RGB в пространство $YC_rC_b$ (первый канал – линейная комбинация интенсивностей по трем каналам исходного изображения, второй/третий – функции разностей интенсивностей красного/синего и полученного значения интенсивности первого канала), **CV_YCrCb2BGR**, **CV_YCrCb2RGB** – обратное конвертирование;
- **CV_BGR2HSV**, **CV_RGB2HSV**, **CV_HSV2BGR**, **CV_HSV2RGB** – коды операций преобразования из RGB/BGR пространств в HSV и наоборот;
- **CV_BGR2HLS**, **CV_RGB2HLS**, **CV_HLS2BGR**, **CV_HLS2RGB** – коды операций конвертирования из RGB/BGR-пространств в HLS и наоборот;
- **CV_BGR2Lab**, **CV_RGB2Lab**, **CV_Lab2BGR**, **CV_Lab2RGB** – коды, обеспечивающие преобразование из BGR/RGB-пространств в CIE L*a*b* и обратно;
- **CV_BGR2Luv**, **CV_RGB2Luv**, **CV_Luv2BGR**, **CV_Luv2RGB** – коды, обеспечивающие преобразование из BGR/RGB пространства в CIE L*u*v* и обратно;
- **CV_BayerBG2BGR**, **CV_BayerGB2BGR**, **CV_BayerRG2BGR**, **CV_BayerGR2BGR**, **CV_BayerBG2RGB**, **CV_BayerGB2RGB**,

**CV_BayerRG2RGB**, **CV_BayerGR2RGB**. Шаблон Байера (Bayer pattern) – это широко используемый формат хранения изображений в CCD и CMOS камерах. Интенсивности по всем каналам хранятся в одной плоскости, конвертирование в RGB/BGR пространство выполняется посредством интерполяции значений интенсивностей пикселей, принадлежащих некоторой окрестности и содержащих интенсивность соответствующего канала.

### 4.2.7. Масштабирование изображения

В данном разделе рассматривается операция масштабирования изображения, которая часто используется в алгоритмах, где требуется построить пирамиду, составленную из разных масштабов одного и того же изображения. Ниже представлен прототип соответствующей функции библиотеки OpenCV.

```
void resize(const Mat& src, Mat& dst, Size dsize,
            double fx=0, double fy=0,
            int interpolation=INTER_LINEAR)
```

Функция **resize** в качестве входных параметров принимает исходное изображение **src**, матрицу **dst**, куда будет записан результат масштабирования, размер **dsize** результирующего изображения, масштабирующие коэффициенты **fx** и **fy** по горизонтали и вертикали соответственно и идентификатор метода интерполяции. Если **dsize=0**, тогда **fx** и **fy** должны быть ненулевыми, и **dsize** вычисляется по формуле **dsize = Size(round(fx*src.cols), round(fy*src.rows))**

Если хотя бы один из масштабирующих коэффициентов принимает нулевое значение, тогда его значение определяется в соответствии с одной из формул в зависимости от направления масштабирования (горизонталь или вертикаль): **fx=(double)dsize.width/src.cols**, **fy=(double)dsize.height/src.rows**.

Разработчики библиотеки поддерживают следующие методы интерполяции:

- **INTER_NEAREST** – интерполяция по методу ближайшего соседа;
- **INTER_LINEAR** – билинейная интерполяция (используется по умолчанию);
- **INTER_AREA** – интерполяция посредством использования отношения пикселей в некоторой области. Приближение изображения показывает, что результат данной интерполяции очень похож на **INTER_NEAREST**;
- **INTER_CUBIC** – бикубическая интерполяция с использованием окрестности 4x4 пикселя;

- **INTER_LANCZOS4** – интерполяция Ланшоца (Lanczos) с окрестностью 8x8 пикселей.

### 4.2.8. Выделение подобласти изображения

Существует значительный класс алгоритмов, которые работают не на всем изображении, а только на некоторой его подобласти – *области интереса* ($region\ of\ interests$). Поэтому на данном этапе рассмотрим функции для выделения такой подобласти. По существу это операция выделения подматрицы в матрице **Mat**. Класс **Mat** имеет два перегруженных оператора круглые скобки, которые и позволяют получить подобласть изображения.

```
Mat operator()( Range rowRange, Range colRange ) const;
Mat operator()( const Rect& roi ) const;
```

Разница между указанными методами лишь в том, что в первом случае на вход отдаются интервалы изменения индексов по горизонтали **rowRange** и по вертикали **colRange**, а во втором – прямоугольник **roi**, который необходимо выделить из изображения.

### 4.2.9. Бинаризация изображения

Остановимся на рассмотрении функции отсечения и бинаризации изображения. Операция отсечения предполагает, что имеется изображение в оттенках серого. В случае бинаризации на выходе формируется черно-белое изображение согласно правилу: если в текущем пикселе исходного изображения интенсивность меньше некоторого порога, то в результирующем изображении данный пиксель имеет черный цвет, в противном случае, белый или наоборот. Операция отсечения является более общей и на выходе может формироваться изображение в оттенках серого, но в основе по-прежнему лежит принцип сравнения с заданным пороговым значением интенсивности.

Функция **threshold** реализует указанную операцию отсечения для изображения **src** по порогу **thresh** и записывает результат в матрицу **dst**.

```
double threshold(const Mat& src, Mat& dst, double thresh,
                 double maxVal, int thresholdType)
```

Рассмотрим возможные значения параметра **thresholdType**, который определяет правило отсечения:

- **THRESH_BINARY**, вычисление интенсивности пикселя с координатами $(x, y)$ осуществляется согласно формуле (по существу реализует операцию бинаризации):

$$dst(x,y) = \begin{cases} maxVal, \text{если } src(x,y) > thresh \\ 0, \text{в противном случае} \end{cases}.$$

- **THRESH_BINARY_INV**:

$$dst(x,y) = \begin{cases} 0, \text{если } src(x,y) > thresh \\ maxVal, \text{в противном случае} \end{cases}.$$

- **THRESH_TRUNC**:

$$dst(x,y) = \begin{cases} thresh, \text{если } src(x,y) > thresh \\ src(x,y), \text{в противном случае} \end{cases}.$$

- **THRESH_TOZERO**:

$$dst(x,y) = \begin{cases} src(x,y), \text{если } src(x,y) > thresh \\ 0, \text{в противном случае} \end{cases}.$$

- **THRESH_TOZERO_INV**:

$$dst(x,y) = \begin{cases} 0, \text{если } src(x,y) > thresh \\ src(x,y), \text{в противном случае} \end{cases}.$$

### 4.2.10. Поиск контуров на бинарном изображении

Рассмотрим функцию **findContours** для определения контуров объектов на бинарном (черно-белом) изображении. Данная функция реализует алгоритм, описанный в работе [21]. Позднее будет разработано приложение, демонстрирующее ее использование. Ниже приведены прототипы соответствующей функции.

```
void findContours(const Mat& image,
                  vector<vector<Point> >& contours,
                  vector<Vec4i>& hierarchy, int mode,
                  int method, Point offset=Point())
void findContours(const Mat& image,
                  vector<vector<Point> >& contours,
                  int mode, int method,
                  Point offset=Point())
```

Функция принимает на вход одноканальное 8-битное изображение **image** и возвращает набор контуров **contours**, каждый из которых задается совокупностью точек. Опционально функция возвращает иерархию отношений между контурами **hierarchy**, для каждого контура **contours[i]** в **hierarchy[i][0]**, **hierarchy[i][1]**, **hierarchy[i][2]**, **hierarchy[i][3]** хранятся индексы следующего и предыдущего контуров того же уровня иерархии, индексы первого дочернего и родительского контуров соответственно. Если какого-либо элемента приведенной последовательности нет, то в соответствующей ячейке хранится отрицательное значение. Наряду с описанными параметрами функция принимает идентификаторы способа восстановления контура **mode** и метода аппроксимации контура **method**. Параметр **mode** может принимать следующие значения:

- **CV_RETR_EXTERNAL** обеспечивает восстановление только внешних контуров, т.е. **hierarchy[i][2]** = **hierarchy[i][3] = -1**;
- **CV_RETR_LIST** позволяет восстанавливать все контуры без установления иерархии;
- **CV_RETR_CCOMP** обеспечивает восстановление всех контуров (внешних и внутренних) и собирает их в двухуровневую иерархию;
- **CV_RETR_TREE** предоставляет возможность получения полной иерархии контуров.

Перечислим возможные значения параметра **method**.

- **CV_CHAIN_APPROX_NONE** – метод, согласно которому хранятся все точки контуров;
- **CV_CHAIN_APPROX_SIMPLE** обеспечивает хранение горизонтальных, вертикальных и диагональных сегментов, сохраняя только их конечные точки;
- **CV_CHAIN_APPROX_TC89_L1**,**CV_CHAIN_APPROX_TC89_KCOS** обеспечивают применение одной из разновидностей алгоритма Тех-Чин (Teh-Chin) [23].

Осталось сказать несколько слов о последнем параметре **offset** функции **findContours**. Данный параметр задает постоянное смещение каждой точки контура. Как правило, используется в случае, если поиск контура выполняется в некоторой области интереса, а отображение контуров необходимо выполнить на полном изображении.

### 4.2.11. Отображение контуров на изображении

На данный момент известно, как получить контуры объектов на изображении, и можно переходить к вопросу, как отобразить полученные контуры. Такой функционал сосредоточен в функции **drawContours**, которая позволяет отобразить на изображении **image** контур **contours[contourIdx]** посредством отрисовки линии, имеющей цвет **color** и толщину **thickness**. Наряду с этим можно указать тип связности линии **lineType** (**8**, **4**, **CV_AA**). Опционально также можно передать иерархию контуров **hierarchy**, максимальное количество уровней иерархии для отображения **maxLevel** и сдвиг каждой точки контура **offset**.

```
void drawContours(Mat& image,
                  const vector<vector<Point> >& contours,
                  int contourIdx, const Scalar& color,
                  int thickness=1, int lineType=8,
                  const vector<Vec4i>&
                          hierarchy=vector<Vec4i>(),
                  int maxLevel=INT_MAX,
```

```
                    Point offset=Point())
```

## 4.3. Разработка приложения для определения контуров объектов

На данном этапе можно переходить к разработке приложения для определения контуров объектов. Сформулируем, что должно быть результатом работы приложения. Во-первых, естественно необходимо видеть изображение, на котором отмечены контуры объектов. Во-вторых, имеет смысл одновременно показывать исходное изображение с целью последующего объяснения результатов.

Сначала создадим решение и проект **01_Contours** согласно схеме, описанной в разделе 3.1, и добавим файл исходного кода, в который поместим основную функцию **`main`**.

Опираясь на рассмотренные ранее операции работы с изображениями, последовательность действий можно представить следующим образом:

1. Загрузить исходное изображение.

```
// загрузка исходного изображения
Mat src = imread("apple.bmp", 1);
if (src.data == 0)
{
      printf("Incorrect image name or format.\n");
      return 1;
}
```

2. Создать копию изображения, т.к. впоследствии необходимо видеть не только изображение с отрисованными контурами, но и исходное.

```
// создание копии исходного изображения
Mat copy = src.clone();
```

3. Конвертировать копию изображения в оттенки серого.

```
Mat gray;
cvtColor(src, gray, CV_BGR2GRAY);
```

4. Выполнить бинаризацию полученного изображения. Операция необходима, т.к. функция поиска контуров работает на черно-белых изображениях.

```
Mat grayThresh;
threshold(gray, grayThresh, 120, 255, CV_THRESH_BINARY);
```

5. Определить контуры на бинарном изображении.

```
// поиск контуров
vector<vector<Point> > contours;
findContours(grayThresh, contours, CV_RETR_CCOMP,
             CV_CHAIN_APPROX_SIMPLE);
```

6. Отобразить исходное изображение и его копию с отрисованными контурами объектов.

```
// создание окон для отображения
const char *srcWinName = "src",
           *contourWinName = "contour";
namedWindow(srcWinName, 1);
namedWindow(contourWinName,1);

// отображение контуров
Scalar color(0, 255, 0);
drawContours(copy, contours, -1, color, 2);

// отображение изображений
imshow(contourWinName, copy);
imshow(srcWinName, src);

// ожидание нажатия какой-либо клавиши
waitKey(0);
```

7. Освободить все занятые ресурсы.

```
// освобождение ресурсов
gray.release();
grayThresh.release();
copy.release();
src.release();
```

## 4.4. Запуск приложения и анализ результатов

Исходный код приложения разработан, поэтому осталось его запустить, нажав, например, комбинацию клавиш **Ctrl+F5**. На рис. 16 показан результат исполнения приложения, полученный на тестовом изображении. Посмотрим внимательно на исходное изображение и полученные контуры, попробуем проинтерпретировать результат.

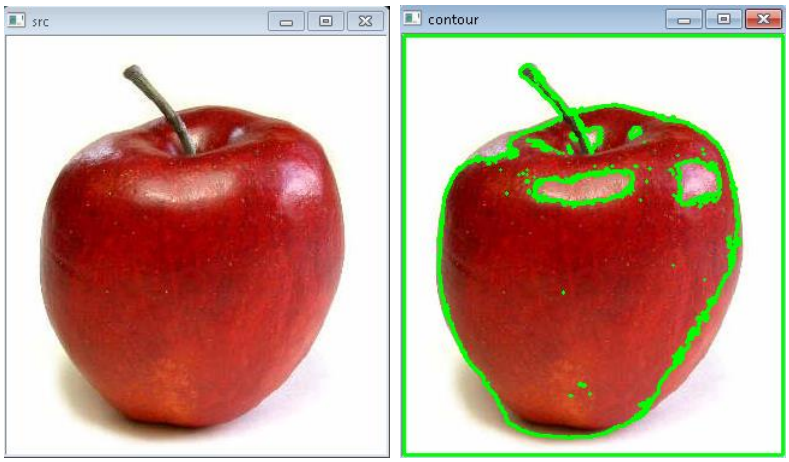

**Рис. 16.** Результат работы приложения для определения контуров объектов

Первый шаг, который выполняется в приложении, – это конвертация исходного изображения в оттенки серого. Результат показан на рис. 17 (слева). Затем выполняется отсечение с порогом интенсивности 120, т.е. для всех пикселей, интенсивность которых меньше 120, в бинарном изображении будет поставлен черный цвет, в противном случае, белый (рис. 17, справа). Уже из бинарного изображения можно приблизительно оценить контуры. Заметим, что блики на объекте воспринимаются как отдельные компоненты, т.к. они соответствуют перепадам интенсивности. Очевидно, что контуры, которые определяются на основании бинарного изображения, отвечают контурам компонент связности.

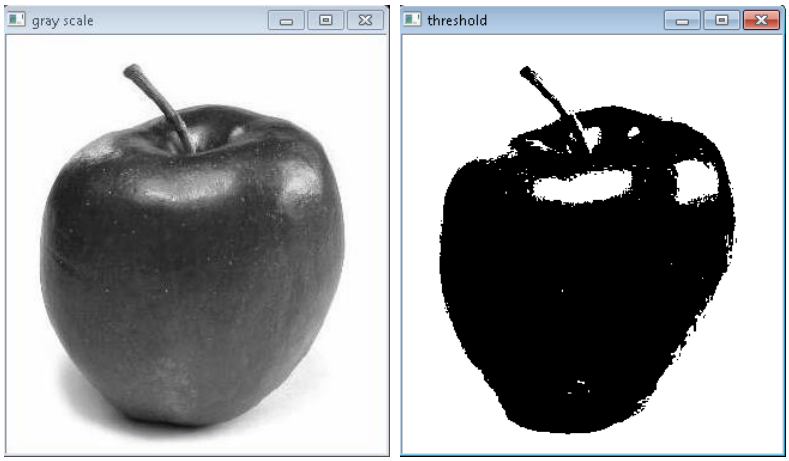


**Рис. 17.** Результаты выполнения промежуточных операций преобразования изображения в оттенки серого (слева) и отсечения (справа)

# 5. Разработка приложения для демонстрации базовых операций работы с видеоданными

## 5.1. Задача детектирования лиц

Задача детектирования лиц (face detection) на изображении во многих случаях является подзадачей при решении более сложных проблем (например, распознавания лица или определения выражения лица). Тем не менее, информация о присутствии лиц и их местоположении может быть полезна для таких приложений, как охранные системы и системы детектирования лиц в фото- и видеотехнике [16].

В зависимости от алгоритма решения задачи детектирования лиц решение представляет собой совокупность прямоугольников либо окружностей, окаймляющих лица.

Задача автоматического определения положения лиц является достаточно сложной, и этому существует несколько объяснений:

1. Значительное расхождение внешнего вида и формы лица.
2. Различие изображения лица при изменении ориентации относительно камеры.
3. Вариативность выражения лица.
4. Возможность перекрытия полного изображения, например, прядью волос или полями шляпы.
5. Другие субъективные факторы такие, как освещенность, искажения, которые вносятся оптикой.

## 5.2. Базовые операции

### 5.2.1. Загрузка видео из файла, перехват видеопотока с камеры

Работа с видео в OpenCV организована посредством использования функционала класса **`VideoCapture`**. Чтобы открыть видеофайл или поток видео с конкретного устройства, достаточно создать объект указанного класса, вызвав конструктор по умолчанию и метод **`open`**, которому на вход передать название файла или идентификатор устройства соответственно.

```
VideoCapture();
virtual bool open(const string& filename);
virtual bool open(int device);
```

Альтернативный вариант – в конструктор класса передать название видеофайла или идентификатор устройства.

```
VideoCapture(const string& filename);
VideoCapture(int device);
```

Для проверки корректности открытия видеопотока, необходимо вызвать метод **`isOpened`**. Прототип метода представлен ниже.

```
virtual bool isOpened() const;
```

После обработки видеопотока необходимо освободить занятые ресурсы посредством вызова метода **`release`**.

```
virtual void release();
```

### 5.2.2. Извлечение кадров видеопотока

Продолжая рассматривать методы работы с видепотоком, обратимся к операции получения следующего кадра. Для этого необходимо последовательно вызвать методы **grab** и **retrieve**, передав второму методу в качестве входного параметра объект типа **Mat**, куда будет записан кадр видео.

```
virtual bool grab();
virtual bool retrieve(Mat& image, int channel=0);
```

Другой вариант получения следующего кадра видеопотока – это использовать перегруженную операцию перенаправления потока ввода. Прототип данной операции приведен ниже.

```
virtual VideoCapture& operator >> (Mat& image);
```

### 5.2.3. Получение и установка свойств видеофайла

В некоторых случаях требуется получить/установить свойства видео. Для этого достаточно использовать методы **get**/**set**, реализованные в классе **VideoCapture**. Метод **set** позволяет установить свойство с идентификатором **propId**, равным значению **value**. Метод **get** возвращает значение свойства **propId**.

```
virtual bool set(int propId, double value);
virtual double get(int propId);
```

Перечислим возможные значения идентификаторов свойств.

- **CV_CAP_PROP_POS_MSEC** – текущая позиция кадра в треке в миллисекундах;
- **CV_CAP_PROP_POS_FRAMES** – индекс следующего кадра, который будет извлекаться из видеопотока (индексация начинается от 0);
- **CV_CAP_PROP_POS_AVI_RATIO** – относительная позиция в видеофайле (0 – начало трека, 1 – конец трека)
- **CV_CAP_PROP_FRAME_WIDTH** – ширина кадра;
- **CV_CAP_PROP_FRAME_HEIGHT** – высота кадра;
- **CV_CAP_PROP_FPS** – количество кадров, отображаемых в секунду;
- **CV_CAP_PROP_FOURCC** – код кодека;
- **CV_CAP_PROP_FRAME_COUNT** – количество кадров в видео;
- **CV_CAP_PROP_FORMAT** – формат объекта типа **Mat**, который возвращается методом **retrieve**;
- **CV_CAP_PROP_MODE** – значение, которое определяет текущий режим захвата;

- **CV_CAP_PROP_BRIGHTNESS** – яркость изображения (только для камер);
- **CV_CAP_PROP_CONTRAST** – контрастность изображения (только для камер);
- **CV_CAP_PROP_SATURATION** – насыщенность изображения (только для камер);
- **CV_CAP_PROP_HUE** – тон изображения (только для камер);
- **CV_CAP_PROP_GAIN** – усиление изображение (только для камер);
- **CV_CAP_PROP_EXPOSURE** – выдержка (только для камер);
- **CV_CAP_PROP_CONVERT_RGB** – булев флаг, который указывает на необходимость конвертации изображения в RGB формат.

### 5.2.4. Сохранение кадров видеопотока в видеофайл

При разработке реальных приложений очень часто возникают ситуации, когда необходимо перехватывать видеопоток с камеры и сохранять его или обработанную копию в видеофайл. Разработчики библиотеки OpenCV обеспечивают данную возможность посредством предоставления класса **VideoWriter**. По структуре и именованию методов класс похож на **VideoCapture**. Чтобы открыть поток на запись можно создать объект указанного класса, используя конструктор по умолчанию, как следствие, для открытия потока необходимо вызвать метод **open**.

```
VideoWriter();
virtual bool open(const string& filename, int fourcc,
                  double fps, Size frameSize,
                  bool isColor=true);
```

Другой вариант – это вызов конструктора с параметрами, который аналогично методу **open** принимает на вход название видеофайла для записи **filename**, идентификатор кодека сжатия **fourcc**, количество кадров в секунду **fps** и размер кадра **frameSize**. Дополнительно присутствует параметр по умолчанию **isColor**, который определяет формат записи (цветное видео или видео в оттенках серого).

```
VideoWriter(const string& filename, int fourcc,
            double fps, Size frameSize, bool isColor=true);
```

По аналогии с интерфейсом класса **VideoCapture** имеется метод **isOpened**, позволяющий определить, открыт ли поток записи или нет.

```
virtual bool isOpened() const;
```

Для записи кадра в видеофайл используется перегруженная операция перенаправления потока вывода.

```
virtual VideoWriter& operator << (const Mat& image);
```

### 5.2.5. Детектирование лиц на изображении с использованием классификатора Хаара

Доступ к реализации каскадного классификатора Хаара осуществляется посредством работы с объектами класса **CascadeClassifier**. В данном разделе рассмотрим некоторые методы указанного класса, которые позволяют выполнять поиск лиц. При этом опустим этап тренировки классификатора, т.к. это достаточно длительная по времени и вычислительно трудоемкая процедура. В состав установленного пакета файлов библиотеки OpenCV уже входят натренированные модели[3].

Для создания объекта классификатора можно использовать конструктор класса по умолчанию, после вызова которого необходимо загрузить натренированную модель лица с помощью метода **load**.

```
CascadeClassifier();
bool load(const string& filename);
```

Альтернативная возможность – создать объект посредством вызова конструктора с параметрами, который явно принимает в качестве входного параметра название файла, содержащего модель.

```
CascadeClassifier(const string& filename);
```

Чтобы найти лица разного масштаба на изображении, необходимо вызвать метод **detectMultiScale**.

```
void detectMultiScale(const Mat& image,
                      vector<Rect>& objects,
                      double scaleFactor=1.1,
                      int minNeighbors=3,
                      int flags=0, Size minSize=Size());
```

Рассмотрим параметры метода. Параметр **image** – это изображение в оттенках серого (типа **CV_8U**), на котором необходимо продетектировать лица и записать окаймляющие прямоугольники в вектор **objects**. **scaleFactor** представляет собой коэффициент масштабирования, который определяет, во сколько раз будет уменьшено изображение на каждом масштабе. Параметр **minNeighbors** является параметром алгоритма и определяет интенсивность обнаружения лица в заданной позиции, что позволяет отсекать одиночные срабатывания детектора, которые с большой вероятностью являются ложными, и объединять в единую сущность срабатывания, близкие по расположению. В текущей реализации классификатора значение **flags** не используется и присутствует для совместимости со старым интерфейсом. Параметр **size** определяет минимальный размер возможного объекта.

[3] Модели можно найти в директории OpenCV-2.4.2\opencv\data\haarcascades.

### 5.2.6. Отображение прямоугольников на изображении

Как было описано в предыдущем разделе, при детектировании лиц с помощью функции **detectMultiScale** возвращается вектор окаймляющих прямоугольников. Поэтому естественно, чтобы увидеть результат детектирования необходимо знать, как отображать эти прямоугольники на изображении.

В общем случае, чтобы отобразить изображение, на котором отрисованы какие-либо геометрические примитивы (прямоугольники, квадраты, окружности и т.п.), необходимо выполнить две операции: нарисовать примитив на изображении и отобразить полученное изображение так, как описано в разделе 4.2.4. Отметим, что функции отрисовки примитивов, как правило, называются в соответствии с англоязычными названиями этих примитивов.

Чтобы нарисовать прямоугольник, достаточно вызвать функцию **rectangle**, которой в качестве входных параметров передать изображения **img**, координаты левого верхнего **pt1** и правого нижнего угла **pt2** прямоугольника, а также цвет линии **color** для RGB-изображения или яркость для изображения в оттенках серого, толщину **thickness** и тип **lineType**. Обратите внимание, что если передать отрицательное значение толщины, что соответствует константе **CV_FILLED**, то будет отрисован прямоугольник, залитый цветом. Параметр **shift** означает количество дробных бит в координатах точек.

```
void rectangle(CvArr* img, CvPoint pt1, CvPoint pt2,
               CvScalar color, int thickness=1,
               int lineType=8, int shift=0)
```

Заметим, что наряду с геометрическими примитивами подобным образом можно написать текст на изображении. Для этого достаточно вызвать функцию **putText**. Подробнее с возможным набором примитивов можно ознакомиться в документации [17].

## 5.3. Разработка приложения для детектирования лиц на видео

Создадим проект **02_FaceDetection** и файл исходного кода **main.cpp** в существующем решении. Конечная цель – разработать консольное приложение, которое позволяет детектировать лица на видеопотоке с использованием классификатора Хаара. Предполагается, что видеопоток извлекается из файла или поступает с веб-камеры. В качестве аргументов командной строки будем передавать название файла, содержащего натренированную модель лица, и при необходимости путь до видеофайла. Также требуется предусмотреть возможность принудительной остановки обработки видео.

```
const char* helper =
```

```
      "02_FaceDetection.exe <model_file> [<video>]\n\
      \t<model_file> - полное имя файла с моделью\n\
      \t<video> - путь до видеофайла (по умолчанию \n\
                 видеопоток принимается с камеры)\n";
```

Создадим функцию **main** с пустым телом. Перед функцией объявим пару констант **DELAY** и **ESC_KEY**. **DELAY** будем использовать, чтобы зафиксировать время задержки при последовательном отображении кадров видео, **ESC_KEY** – чтобы обозначить код клавиши **Esc**, при нажатии которой будет происходить принудительная остановка обработки видеопотока.

```
#define DELAY 30
#define ESC_KEY 27
```

Обратимся к разработке кода основной функции. Для этого необходимо реализовать следующую последовательность операций:

1. Разобрать аргументы командной строки, чтобы извлечь полное название файла с моделью лица и при необходимости имя видеофайла.

```
char *modelFileName = 0, *videoFileName = 0;
if (argc < 2)
{
      printf("%s", helper);
      return 1;
}
modelFileName = argv[1];
if (argc > 2)
{
      videoFileName = argv[2];
}
```

2. Создать классификатор и загрузить модель.

```
// создание классификатора и загрузка модели
CascadeClassifier cascade;
cascade.load(modelFileName);
```

3. Загрузить видео из файла или открыть видеопоток с веб-камеры, если название файла отсутствует в параметрах командной строки.

```
// загрузка видеофайла или перехват видеопотока
VideoCapture capture;
if (videoFileName == 0)
{
      capture.open(0);
}
else
{
      capture.open(videoFileName);
}
if (!capture.isOpened())
```

```
{
      printf("Incorrect capture name.\n");
      return 1;
}
```

4. Создать окно для отображения видео.

```
const char* winName = "video";
// создание окна для отображения видео
namedWindow(winName);
```

5. Разработать цикл обработки последовательности кадров видепотока. На каждой итерации цикла необходимо конвертировать текущий кадр в оттенки серого, продетектировать лица, отрисовать полученные окаймляющие прямоугольники на изображении и отобразить это изображение. Тело цикла будет выполняться до тех пор, пока не будет достигнут конец трека, либо не будет нажата клавиша **Esc**.

```
char key = -1;
Mat image, gray;
vector<Rect> objects;
// получение кадра видеопотока
capture >> image;
while (image.data != 0 && key != ESC_KEY)
{
      cvtColor(image, gray, CV_BGR2GRAY);
      cascade.detectMultiScale(gray, objects);
      for (i = 0; i < objects.size(); i++)
      {
            rectangle(image,
                Point(objects[i].x, objects[i].y),
                Point(objects[i].x+objects[i].width,
                     objects[i].y+objects[i].height),
                CV_RGB(255, 0, 0), 2);
      }
      imshow(winName, image);
      key = waitKey(DELAY);
      capture >> image;
      gray.release();
      objects.clear();
}
```

6. Освободить ресурсы и закрыть видеопоток.

```
capture.release();
```

## 5.4. Запуск приложения и анализ результатов

Сделаем проект **02_FaceDetection** рабочим, выполнив команду контекстного меню **Set as StartUp Project…**. Скомпилируем и запустим приложение. Будем считать, что видеопоток принимается с веб-камеры,

поэтому передадим в качестве параметра командной строки только файл модели. Используем файл **haarcascade_frontalface_default.xml**.

На рис. 18 показан результат детектирования на одном из кадров видеопотока. Приведенное изображение демонстрирует идеальный случай, когда классификатор достаточно точно обнаружил лицо человека. В данном случае неплохой результат во многом обусловлен наличием однородного фона и отсутствием предметов со сложным цветом и текстурой. Тем не менее, среди кадров даже «простого» трека всегда можно найти такие, где классификатор ошибается, т.е. вообще не находит лицо либо считает отдельные части изображения лицами.


**Рис. 18.** Результат детектирования лиц на одном из кадров видеопотока

В качестве дополнительных заданий читателю предлагается поэкспериментировать со сценами, содержащими группы людей, посмотреть результаты детектирования лиц и проанализировать полученные результаты.

## 6. Контрольные вопросы

1. Перечислите основные способы установки библиотеки OpenCV, их преимущества и недостатки.
2. Назначение утилиты CMake. Причины использования данной утилиты в процессе разработки крупных проектов.
3. Приведите примеры сложных задач, в которых возникает задача детектирования контуров в качестве подзадачи.
4. Приведите примеры систем, в которых используется решение задачи детектирования лиц.

# 7. Дополнительные задания

1. Модифицируйте приложение для поиска контуров объекта так, чтобы результирующее изображение сохранялось в файл.
2. Модифицируйте приложение для поиска контуров объекта так, чтобы отображалось изображение, конвертированное в оттенки серого, и бинаризованное изображение, т.е. то, что показано на рис. 17.
3. Как будет изменяться результат работы приложения для определения контуров, если изменить способ восстановления контуров? Проведите эксперименты со всеми возможными значениями параметра **`mode`** в функции **`findContours`**.
4. Модифицируйте приложения, которое выполняет детектирование лиц на видеопотоке, так, чтобы результат детектирования сохранялся в видеофайл. Примечание: используйте возможности класса **`VideoWriter`**.
5. Снимите видео, где была бы группа людей с разным ракурсом лица. Проанализируйте результаты детектирования на данном видео. Сравните результаты детектирования лиц с использованием других моделей лиц. Модифицируйте приложение так, чтобы добиться максимального качества детектирования. Примечание: комбинируйте результаты детектирования с помощью моделей фаса и профиля.
6. Добавьте в приложение для детектирования лиц возможность детектирования глаз и других частей лица с использованием классификатора Хаара. Примечание: используйте натренированные модели, входящие в состав библиотеки OpenCV.

# 8. Литература

## 8.1. Основная литература

1. Bradski G., Kaehler A. Learning OpenCV. – O'Reilly, 2008. – 571p.

## 8.2. Ресурсы сети Интернет

2. Официальная страница библиотеки OpenCV [http://opencv.org/].
3. Официальная страница EmguCV [http://www.emgu.com/wiki/index.php/Main_Page].
4. Официальная страница CMake [http://www.cmake.org/].

5. Сервер для загрузки последней версии исходных кодов библиотеки OpenCV [http://code.opencv.org/svn/opencv/trunk/opencv].
6. Сервер для загрузки последней версии исходных кодов библиотеки OpenCV и дополнительных файлов (например, натренированные модели, файлы входных данных и результатов для автоматических тестов) [http://code.opencv.org/svn/opencv/trunk].
7. Страница загрузки клиента Subversion [http://sourceforge.net/projects/win32svn/files/latest/download].
8. Руководство по сборке и установке библиотеки OpenCV [http://opencv.willowgarage.com/wiki/InstallGuide/].
9. Официальная страница проекта Jasper [http://www.ece.uvic.ca/~frodo/jasper/].
10. Официальная страница проекта libjpeg [http://libjpeg.sourceforge.net/].
11. Официальная страница проекта libpng [http://www.libpng.org/pub/png/libpng.html].
12. Официальная страница проекта libtiff [http://www.libtiff.org/].
13. Официальная страница библиотеки zlib [http://zlib.net/].
14. Описание базовых структур данных библиотеки OpenCV [http://opencv.willowgarage.com/documentation/cpp/basic_structures.html].
15. Описание функций преобразования изображений [http://opencv.willowgarage.com/documentation/cpp/miscellaneous_image_transformations.html].
16. Обнаружение и локализация лица на изображении [http://cgm.computergraphics.ru/content/view/40].
17. Описание функций отрисовки примитивов на изображении [http://opencv.willowgarage.com/documentation/drawing_functions.html].

## 8.3. Дополнительная литература

18. Viola P., Jones M.J. Robust Real-Time Face Detection // international Journal of Computer Vision 57(2). – 2004. – pp. 137-154.
19. Felzenszwalb P. F., Girshick R. B., McAllester D., Ramanan D. Object Detection with Discriminatively Trained Part Based Models // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2010. Vol.32, No.9. – pp. 1627–1645.
20. Форсайт Д., Понс Ж. Компьютерное зрение. Современный подход. – М.: Изд. д. Вильямс, 2004. – 465с.

21. Suzuki S., Abe K. Topological Structural Analysis of Digitized Binary Images by Border Following // CVGIP 30 1. – 1985. – pp. 32-46.

22. Шапиро Л., Стокман Дж. Компьютерное зрение. – М.: Бином. Лаборатория знаний, 2006. – 752с.

23. Teh, Cho-Huak, Chin, Roland T. On the detection of dominant points on digital curves // IEEE Transactions on Pattern Analysis and Machine Intelligence. – Vol. II No. 8. – 1989. – pp. 859-872.

# 9. Приложения

## 9.1. Приложение А. Исходный код программы для поиска контуров на изображении

```
#include <opencv2/opencv.hpp>
using namespace cv;

int main()
{
      // создание окон для отображения
      const char *srcWinName = "src",
                 *contourWinName = "contour";
      namedWindow(srcWinName, 1);
      namedWindow(contourWinName,1);
      // загрузка исходного изображения
      Mat src = imread("apple.bmp", 1);
      if (src.data == 0)
      {
            printf("Incorrect image name or format.\n");
            return 1;
      }
      // создание копии исходного изображения
      Mat copy = src.clone();
      // создание одноканального изображения для
      // конвертирования исходного изображения в
      // оттенки серого
      Mat gray, grayThresh;
      cvtColor(src, gray, CV_BGR2GRAY);
      threshold(gray, grayThresh, 120,
                255, CV_THRESH_BINARY);
      // поиск контуров
      vector<vector<Point> > contours;
      findContours(grayThresh, contours, CV RETR CCOMP,
                   CV_CHAIN_APPROX_SIMPLE);
      // отображение контуров
      Scalar color(0, 255, 0);
      drawContours(copy, contours, -1, color, 2);
```

```
	// отображение изображений
	imshow(contourWinName, copy);
	imshow(srcWinName, src);
	// ожидание нажатия какой-либо клавиши
	waitKey(0);
	// освобождение ресурсов
	gray.release();
	grayThresh.release();
	copy.release();
	src.release();
	return 0;
}
```

## 9.2. Приложение Б. Исходный код приложения для детектирования лиц на видепотоке

```
#include <opencv2/opencv.hpp>
#define DELAY 30
#define ESC_KEY 27

using namespace cv;

const char* helper =
	"02_FaceDetection.exe <model_file> [<video>]\n\
	\t<model_file> - model file name\n\
	\t<video> - video file name (video stream will \n\
	           be taken from web-camera by default)\n\
	";

int main(int argc, char *argv[])
{
	const char* winName = "video";
	char *modelFileName = 0, *videoFileName = 0;
	int i;
	char key = -1;
	Mat image, gray;
	VideoCapture capture;
	vector<Rect> objects;

	if (argc < 2)
	{
		printf("%s", helper);
		return 1;
	}
	modelFileName = argv[1];
	if (argc > 2)
	{
		videoFileName = argv[2];
	}
```

```
	// создание классификатора и загрузка модели
	CascadeClassifier cascade;
	cascade.load(modelFileName);

	// загрузка видеофайла или перехват видеопотока
	if (videoFileName == 0)
	{
		capture.open(0);
	}
	else
	{
		capture.open(videoFileName);
	}
	if (!capture.isOpened())
	{
		printf("Incorrect capture name.\n");
		return 1;
	}

	// создание окна для отображения видео
	namedWindow(winName);

	// получение кадра видеопотока
	capture >> image;
	while (image.data != 0 && key != ESC_KEY)
	{
		cvtColor(image, gray, CV_BGR2GRAY);
		cascade.detectMultiScale(gray, objects);
		for (i = 0; i < objects.size(); i++)
		{
			rectangle(image,
			   Point(objects[i].x, objects[i].y),
			   Point(objects[i].x+objects[i].width,
				objects[i].y+objects[i].height),
			   CV_RGB(255, 0, 0), 2);
		}
		imshow(winName, image);
		key = waitKey(DELAY);
		capture >> image;
		gray.release();
		objects.clear();
	}
	capture.release();
	return 0;
}
```