Николай Прохоренок

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

OpenCV и Java

ОБРАБОТКА ИЗОБРАЖЕНИЙ И КОМПЬЮТЕРНОЕ ЗРЕНИЕ

Загрузка изображения из файла

Захват кадров с веб-камеры

Трансформация изображения

Применение фильтров

Сегментация изображения

Выделение границ объектов

Поиск и сравнение контуров

Поиск объекта по цвету или шаблону

Поиск и сравнение особых точек

Материалы на www.bhv.ru

bhv®

Николай Прохоренок

OpenCV и Java

ОБРАБОТКА ИЗОБРАЖЕНИЙ И КОМПЬЮТЕРНОЕ ЗРЕНИЕ

Санкт-Петербург
«БХВ-Петербург»
2018

УДК 004.93
ББК 32.973.26-018.1
П84

Прохоренок Н. А.

П84 OpenCV и Java. Обработка изображений и компьютерное зрение. — СПб.: БХВ-Петербург, 2018. — 320 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-3955-5

Книга знакомит с современными технологиями компьютерного зрения, позволяющими машинам, роботам, веб-камерам и другим устройствам распознавать изображения. Приведено описание библиотеки компьютерного зрения OpenCV применительно к языку программирования Java. Объясняется, как загружать и сохранять изображения в различных форматах, захватывать кадры с веб-камеры в режиме реального времени, выполнять обработку, трансформацию и сегментацию изображения, применять к изображению фильтры. На практических примерах рассмотрены алгоритмы компьютерного зрения, предназначенные для обнаружения, классификации и отслеживания объектов, выделения границ и контуров объектов, поиска объектов по шаблону, особым точкам, цвету или обученному классификатору.

Для программистов

УДК 004.93
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Марины Дамбиевой*

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-5-9775-3955-5

Оглавление

Введение

Добро пожаловать в мир OpenCV!

OpenCV (от англ. Open Source Computer Vision Library) — это библиотека алгоритмов компьютерного зрения с открытым исходным кодом. Библиотека распространяется по лицензии BSD, следовательно, она может свободно использоваться в академических и коммерческих целях. За долгое время своего существования библиотека приобрела обширную аудиторию пользователей и на сегодняшний день OpenCV является стандартом в области компьютерного зрения. Библиотека написана на языках C/C++, имеет привязки к языкам Python, Java, Matlab и др. В этой книге мы рассмотрим привязку OpenCV к языку программирования Java SE (*SE*, Standard Edition) применительно к операционной системе Windows.

Что же такое компьютерное зрение? *Компьютерное зрение* — это теория и технология получения информации из изображений. Причем изображение может быть как отдельной фотографией, так и последовательностью кадров видео, полученной из видеофайла или с видеокамеры (например, с камеры наружного наблюдения, с веб-камеры или со стереокамеры) в режиме реального времени.

Типичными задачами компьютерного зрения являются обнаружение, отслеживание и классификация объектов. Цель *обнаружения* — найти на изображении объект. Для этого мы можем использовать границы (контуры), особые точки (например, углы), информацию о цвете и т. д. *Отслеживание* применяется в работе с камерами наружного наблюдения. При этом также можно воспользоваться контурами, особыми точками и информацией о цвете, а можно и вычесть фон из текущего кадра (при условии, что камера статична). Чтобы выполнить *классификацию*, нужно распознать обнаруженный объект. Для распознавания можно выполнить попиксельное сравнение с шаблоном, сравнить контуры или особые точки, осуществить поиск по обученному классификатору (например, с помощью каскадов Хаара) и др. В последнее время для распознавания объектов все чаще используются глубокие сверточные нейронные сети. Поддержка таких сетей была добавлена в OpenCV в версии 3.3.0.

С компьютерным зрением тесно связана еще одна технология — *обработка изображения*. Действительно, прежде чем искать объекты на изображении, необходимо выполнить обработку. В частности, в большинстве случаев надо изменить раз-

меры изображения и выполнить выравнивание гистограммы. Кроме того, нужно сгладить изображение, чтобы избавиться от цифрового шума, который возникает при использовании дешевых камер, а также при различных специфических режимах съемки (например, при высоких значениях ISO или при длительных выдержках). Многие алгоритмы очень чувствительны к шумам, и их наличие может привести, например, к неправильно найденным границам объекта или ложным особым точкам.

Ученые занимаются исследованиями в области компьютерного зрения уже очень давно, но до сих пор так и не приблизились к способностям человека распознавать образы. В чем же проблемы? Почему компьютер может с легкостью сложить очень большие числа (что человеку часто не под силу), а вот распознать образ на изображении не в состоянии (человек, заметьте, справляется с этим без проблем)?

Первая проблема заключается в том, что цвет пиксела в цифровом изображении задается числовыми значениями. Причем этих чисел в полноцветных изображениях три: первое число задает количество красного цвета, второе — зеленого и третье — синего (цветовая модель RGB). Вторая проблема состоит в том, что пиксел имеет квадратную форму, а в видео даже встречается прямоугольная форма элементов изображения. Попробуйте нарисовать прямую линию под углом, увеличить масштаб и посмотреть, что получилось. При увеличенном масштабе даже человек не сможет сказать, что это линия. Третья проблема возникает при сжатии изображения, которое используется для уменьшения размера файла. Например, при сохранении в формате JPEG появляются артефакты сжатия, с которыми нужно как-то бороться.

Существует множество и других сложностей. Например, при работе с изображениями мы имеем дело с двумерным пространством, а человек, имея два глаза, видит трехмерную картинку. Попробуйте закрыть один глаз и попасть вилкой в розетку. Согласитесь, что с двумя открытыми глазами получается гораздо лучше. При распознавании образов человек воспринимает не только контуры, точки или символы, но и контекст сцены или смысл текста в целом. Например, попробуйте прочитать этот текст:

```
Не иеемт занчнеия, в кокам пряокде рсапожолены бкувы в солве!
```

Согласитесь, что у вас не возникло никаких проблем с понятием смысла, при этом вы не распознавали буквы внутри каждого слова, иначе бы ничего не поняли.

Еще одна проблема заключается в изменчивости формы объекта. Посмотрим, например, на человека. Он может стоять, сидеть, лежать, нагнуться вперед, назад или в бок и т. д. Поскольку мы имеем дело с двумерными изображениями, то части тела могут перекрываться другими частями тела или другими объектами. Человек может быть одет в разную одежду, носить очки, иметь усы и бороду и т. д. Все это очень сильно осложняет процесс распознавания образов.

Если мы обратимся к обитателям дикой природы, то заметим, что для сохранения собственной жизни животные маскируются под окружающую обстановку. Попробуйте отличить палочника от ветки дерева. Кроме того, животные имеют различную окраску, которая очень затрудняет отделение одного животного от другого, —

например, полосы у зебр. Так что не все так просто, как может показаться на первый взгляд.

В библиотеке OpenCV реализовано очень большое количество алгоритмов для обработки изображений и компьютерного зрения. Некоторые алгоритмы реализованы в виде отдельных классов, а большинство — в виде статических методов какого-либо класса. Классы в OpenCV версии 3.3.0 в Java разделены на следующие пакеты:

- `org.opencv.imgcodecs` — включает класс `Imgcodecs`, с помощью которого можно загрузить изображение из файла или буфера, а также сохранить изображение в файл или в буфер в различных форматах (например, в JPEG);
- `org.opencv.core` — содержит основные классы библиотеки, реализующие базовые структуры (векторы, матрицы и т. д.). Кроме того, пакет включает вспомогательные классы (например, классы `Point`, `Rect` и др.). Класс `Core` из этого пакета содержит статические методы, с помощью которых можно выполнить различные операции с матрицами;
- `org.opencv.imgproc` — включает классы, предназначенные для обработки и анализа изображений;
- `org.opencv.features2d` — содержит классы, с помощью которых можно находить и сравнивать особые точки;
- `org.opencv.photo` — включает классы, предназначенные для создания HDR-изображений;
- `org.opencv.video` — содержит классы, предназначенные для работы с видеоданными (анализ движения и отслеживание объектов);
- `org.opencv.videoio` — с помощью класса `VideoCapture` из этого пакета можно загружать кадры из видеофайла или последовательности кадров, а также захватывать кадры в режиме реального времени с камер наружного видеонаблюдения, веб-камер и др.;
- `org.opencv.calib3d` — содержит классы, с помощью которых можно выполнить калибровку камеры, работать со стереокамерами и обрабатывать трехмерные данные;
- `org.opencv.objdetect` — включает классы для поиска объектов на изображении. С помощью обученных классификаторов можно искать людей, лица, глаза, нос, узнать, улыбается человек или нет, и т. д. При большом желании можно обучить собственный классификатор с произвольным назначением или загрузить уже обученный из Интернета;
- `org.opencv.ml` — содержит классы, предназначенные для машинного обучения;
- `org.opencv.dnn` — включает классы для работы с нейронными сетями. Можно загружать модели, обученные в популярных библиотеках Caffe, TensorFlow и Torch;
- `org.opencv.utils` — содержит вспомогательный класс `Converters`, который в основном используется для внутренних нужд библиотеки;

- `org.opencv.android` — включает классы для работы с ОС Android. Пакет доступен только в составе дистрибутива под Android.

В этой книге мы рассмотрим большинство классов и методов, предназначенных для обработки изображений и компьютерного зрения. При изучении материала от вас потребуется знание основ языка Java SE 8, а также математики на уровне средней школы. Сами алгоритмы мы рассматривать не станем, т. к., во-первых, исходный код библиотеки доступен для просмотра, во-вторых, частичное описание алгоритмов есть в документации, в-третьих, в документации есть ссылки на оригинальные научные статьи, которые и нужно изучать, если возникнет в этом необходимость. И, наконец, описание алгоритмов потребует для книги очень большого объема, а от вас очень глубокого знания математики. Эта книга для всех, поэтому мы будем учиться пользоваться уже реализованными алгоритмами.

Параллельно с изучением книги советую посмотреть два видеокурса по компьютерному зрению, чтобы у вас сложилось более целостное представление о предметной области. Первый видеокурс доступен на канале Антона Конушина:

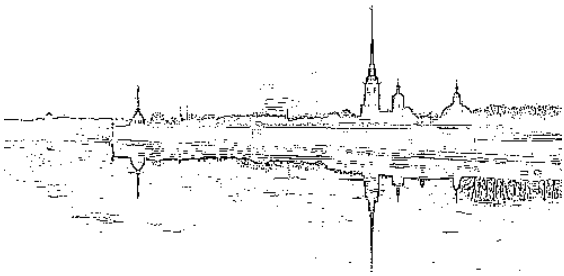
https://www.youtube.com/channel/UC9qIqkpLGA4xg6Nz6BE4aRA

Второй видеокурс доступен на канале Дмитрия Полевого:

https://www.youtube.com/channel/UChNX3pI3bTHLNge294F7yDA

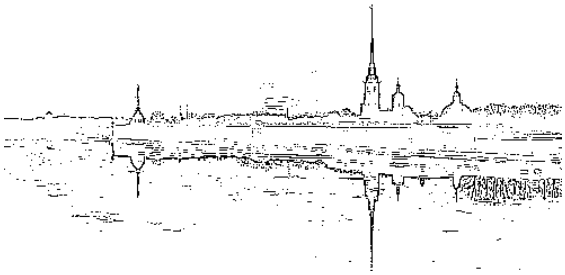
Обратите внимание: на каналах есть различные версии видеокурсов, отличающиеся годом выхода. Советую вам посмотреть их все, чтобы увидеть тенденции развития компьютерного зрения.

Все листинги из этой книги вы найдете в файле Listings.doc, электронный архив с которым можно загрузить с FTP-сервера издательства «БХВ-Петербург» по ссылке: **ftp://ftp.bhv.ru/9785977539555.zip** или со страницы книги на сайте **www.bhv.ru** (см. *приложение*).



ЧАСТЬ I

Загрузка изображения



ГЛАВА 1

Первые шаги

Прежде чем мы начнем рассматривать возможности библиотеки OpenCV, необходимо сделать одно замечание. Не забывайте, что книги по программированию нужно не только читать, но и выполнять все приведенные в них примеры, а также экспериментировать, что-либо в этих примерах изменяя. Поэтому, если вы удобно устроились на диване и настроились просто читать, у вас практически нет шансов изучить предлагаемую вам технологию! Материал в книге изложен очень компактно, без «воды», поэтому просто чтение вам быстро наскучит, и вы станете бегло просматривать листинги, так ничего из них и не усваивая. Однако, если вы сядете перед компьютером и будете не только читать книгу, но и запускать на выполнение все примеры, то процесс окажется увлекательным и запоминающимся, — ведь вы увидите результаты выполнения, но перед этим, наверняка, допустите множество ошибок (например, забудете импортировать класс или импортируете его из другого пакета). Обязательно найдите и исправьте все свои ошибки. Именно поиск и исправление ошибок научат вас очень многому и не дадут вам скучать. Чем больше вы будете делать самостоятельно, тем большему научитесь. Наберитесь терпения, и вперед!

1.1. Установка OpenCV

Вначале нужно скачать и установить библиотеку OpenCV. Для этого открываем в веб-браузере страницу **http://opencv.org/releases.html** и переходим по ссылке **Win pack** из раздела **3.3.0**. Сохраняем файл opencv-3.3.0-vc14.exe на рабочем столе и запускаем его на выполнение. Скачанный нами файл представляет собой архив, а не программу установки, поэтому открывшееся окно (рис. 1.1) просто предлагает вам выбрать место для его распаковки, — нажимаем кнопку **Extract** и распаковываем архив прямо на рабочий стол. Процесс распаковки архива показан на рис. 1.2.

Далее заходим в созданную распаковщиком папку и переименовываем вложенную папку opencv в opencv_3_3, а затем вырезаем эту папку со всем содержимым и вставляем ее в корень диска C или любого другого. В результате путь до файла opencv-330.jar должен стать таким: C:\opencv_3_3\build\java. Если вы решили выбрать другое

место, то помните, что в пути не должно быть русских букв, поэтому корень диска — самое лучшее место для установки библиотеки.

В OpenCV версии 3 некоторые популярные алгоритмы защищены патентами, поэтому они вынесены в отдельный (не свободный) модуль. Эти алгоритмы можно использовать в образовательных целях, но коммерческое их применение требует оплаты. Модуль с защищенными алгоритмами по умолчанию не доступен в Java, однако эти алгоритмы доступны в OpenCV версии 2.4.13. Считаю необходимым представить вам и эти алгоритмы, поэтому некоторые примеры в книге будут написаны для версии OpenCV 2.4.13. Поэтому советую дополнительно установить эту версию в папку C:\opencv_2_4. Путь до файла opencv-2413.jar должен быть таким: C:\opencv_2_4\build\java.

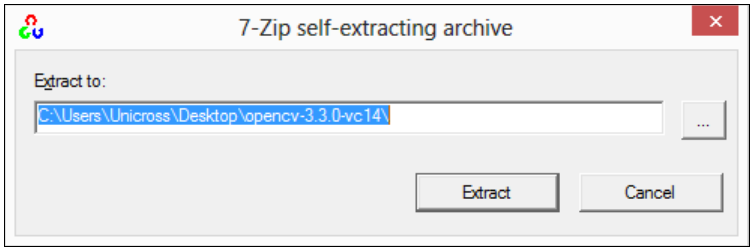


Рис. 1.1. Выбор места установки OpenCV

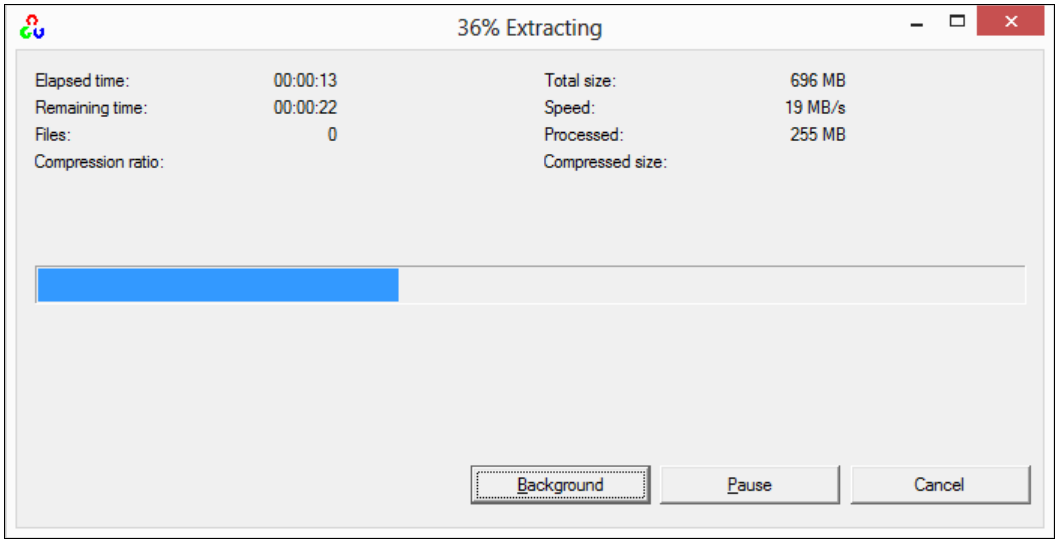


Рис. 1.2. Процесс распаковки архива

1.2. Настройка редактора Eclipse

Для компиляции и запуска примеров из книги мы будем пользоваться редактором Eclipse. Для загрузки редактора Eclipse переходим на страницу **http://www.eclipse.org/downloads/** и скачиваем архив с программой из раздела **Eclipse IDE for Java Developers**. Распаковываем архив в какую-либо папку (в моем случае программа установлена в папку C:\Android\Program) — в результате будет создана папка eclipse, внутри которой расположены файлы редактора. Редактор не нуждается в установке, поэтому просто запускаем файл eclipse.exe. При запуске редактор попросит указать папку с рабочим пространством. Указываем папку и нажимаем кнопку **OK**.

Для отображения результатов выполнения программ мы воспользуемся возможностями библиотеки JavaFX, поэтому дополнительно следует установить модуль e(fx)clipse. Информацию о его установке можно найти на странице **http://www.eclipse.org/efxclipse/install.html**. Прежде чем устанавливать модуль, убедитесь, что на компьютере установлена версия Java SE 8 (причем JDK, а не просто JRE). Если установлена более ранняя версия, то для отображения результатов нужно будет использовать библиотеку `Swing`.

Для создания *проекта* в меню **File** редактора Eclipse выбираем пункт **New | Java Project**. В открывшемся окне (рис. 1.3) в поле **Project name** вводим `OpenCV33`, выбираем версию JRE и нажимаем кнопку **Finish**.

Далее создадим *пакет*. Для этого в меню **File** выбираем пункт **New | Package**. В открывшемся окне (рис. 1.4) в поле **Name** вводим `application` и нажимаем кнопку **Finish**.

Теперь можно добавить в пакет *класс*. Для этого в меню **File** выбираем пункт **New | Class**. В открывшемся окне (рис. 1.5) в поле **Name** вводим `Main`, в поле **Package** — `application`, а затем нажимаем кнопку **Finish**.

Для удобного подключения библиотеки OpenCV к нескольким проектам в меню **Window** выбираем пункт **Preferences**. В открывшемся окне (рис. 1.6) переходим на вкладку **Java | Build Path | User Libraries** и нажимаем кнопку **New**. В открывшемся окне (рис. 1.7) в поле **User library name** вводим `opencv_330` и нажимаем кнопку **OK** — в список будет добавлен новый пункт. Выделяем его, нажимаем кнопку **Add External JARs**, выбираем файл C:\opencv_3_3\build\java\opencv-330.jar и нажимаем кнопку **Открыть** — в списке отобразится новый пункт с названием JAR-архива. Выделяем пункт **Native library location** и нажимаем кнопку **Edit**. В открывшемся окне (рис. 1.8) вводим путь к папке C:\opencv_3_3\build\java\x64 (содержит файл opencv_java330.dll) и нажимаем кнопку **OK**.

Если на вашем компьютере установлена 32-разрядная операционная система или JDK, то нужно будет указать путь к папке C:\opencv_3_3\build\java\x86. Точно таким же образом можно добавить поддержку и для версии 2.4 (JAR-архив opencv-2413.jar и путь C:\opencv_2_4\build\java\x64). Сохраняем изменения, нажимая кнопку **OK**.

Теперь необходимо подключить к проекту библиотеку. Для этого в окне **Package Explorer** щелкаем правой кнопкой мыши на названии проекта и из контекстного

меню выбираем пункт **Properties**. В открывшемся окне (рис. 1.9) переходим на вкладку **Java Build Path**, а затем справа выбираем вкладку **Libraries** и нажимаем кнопку **Add Library**. В открывшемся окне (рис. 1.10) выбираем пункт **User Library** и нажимаем кнопку **Next**. На следующем шаге (рис. 1.11) устанавливаем флажок **opencv_330** и нажимаем кнопку **Finish** — библиотека будет добавлена в список. Сохраняем изменения, нажимая кнопку **OK**. Точно таким же образом создайте проект с названием `OpenCV24` и подключите к нему библиотеку OpenCV версии 2.4.13.

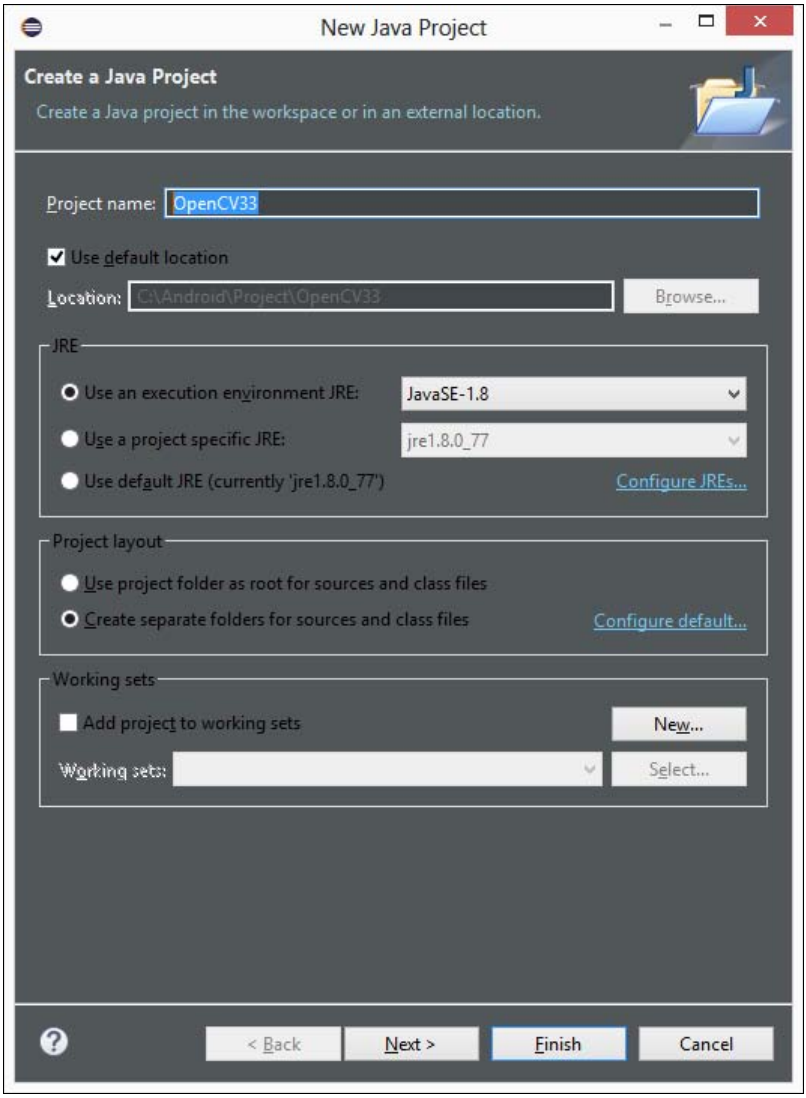


Рис. 1.3. Создание проекта

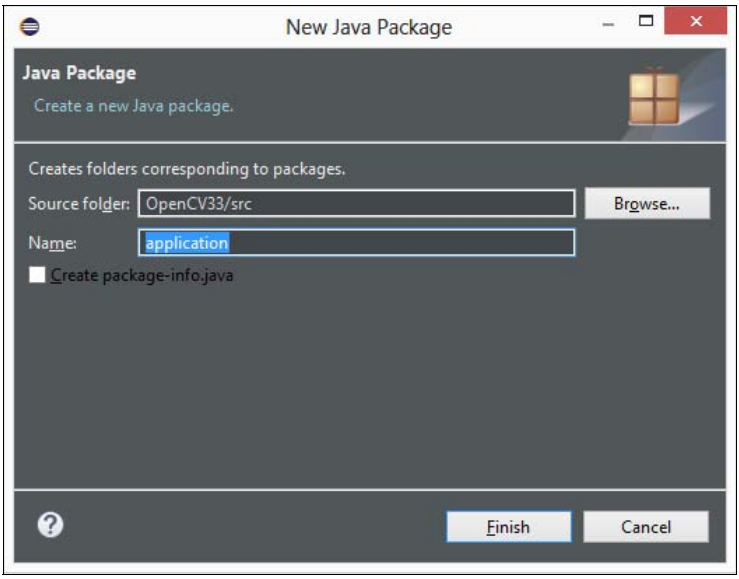


Рис. 1.4. Создание пакета

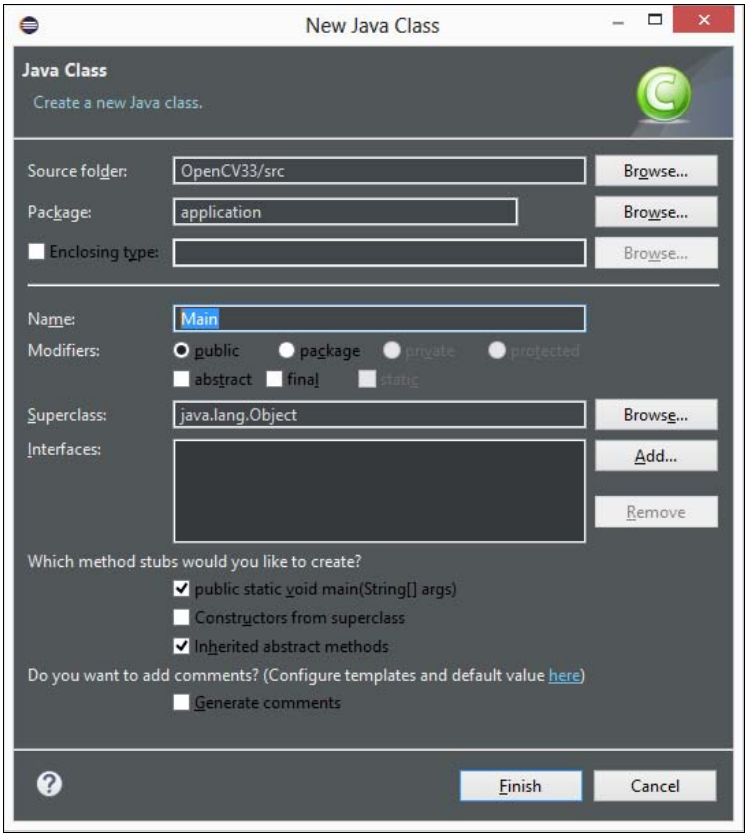


Рис. 1.5. Создание класса

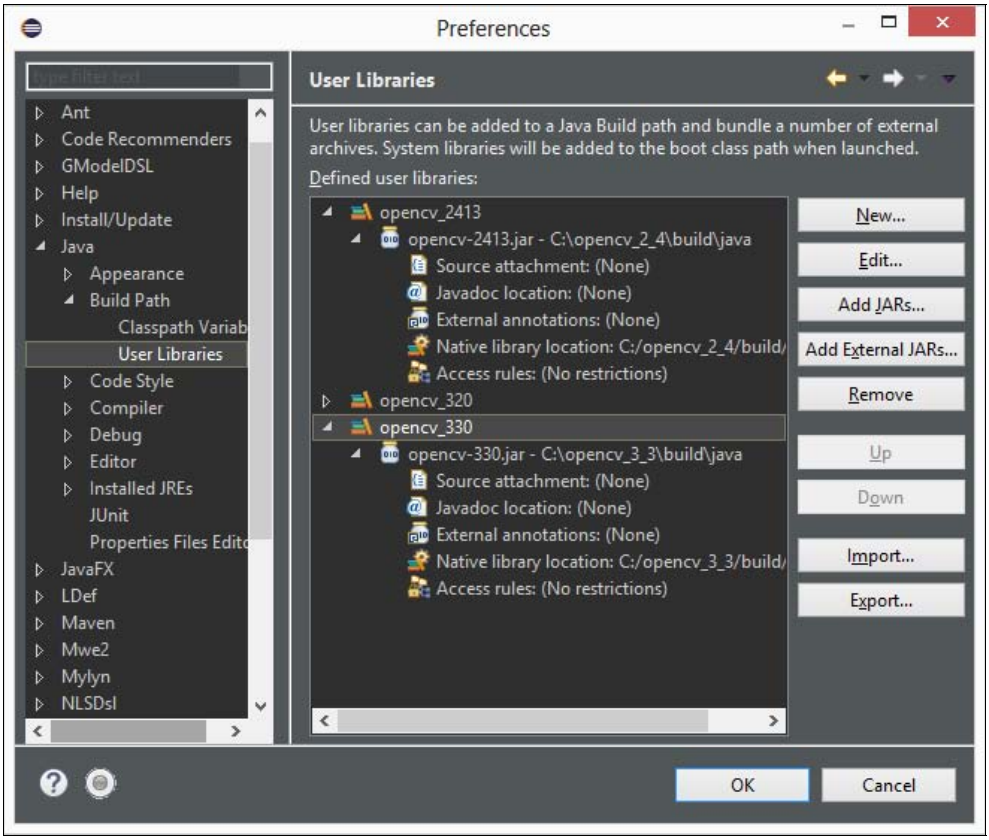


Рис. 1.6. Окно **Preferences**

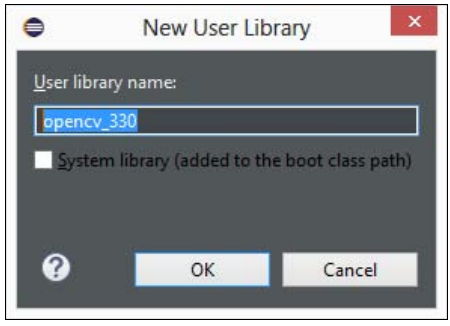


Рис. 1.7. Окно **New User Library**

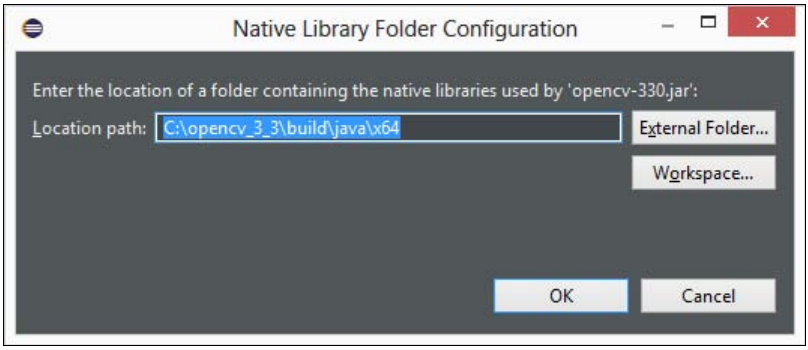


Рис. 1.8. Указание пути к DLL-файлу

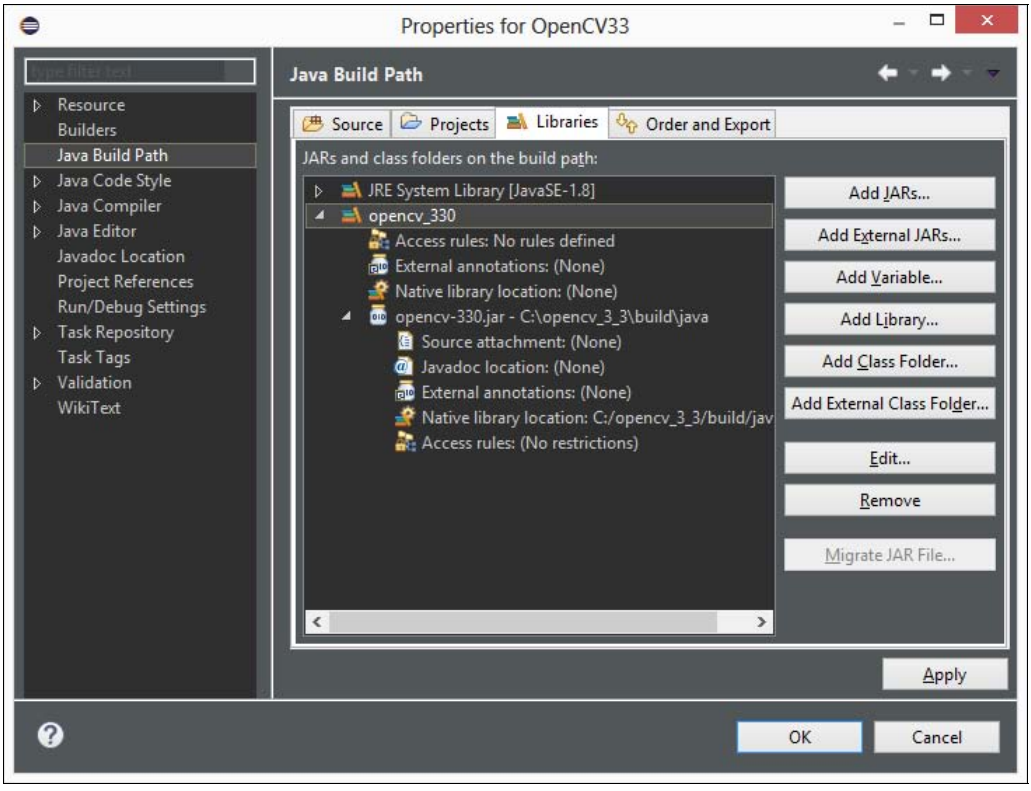


Рис. 1.9. Добавление библиотеки OpenCV к проекту

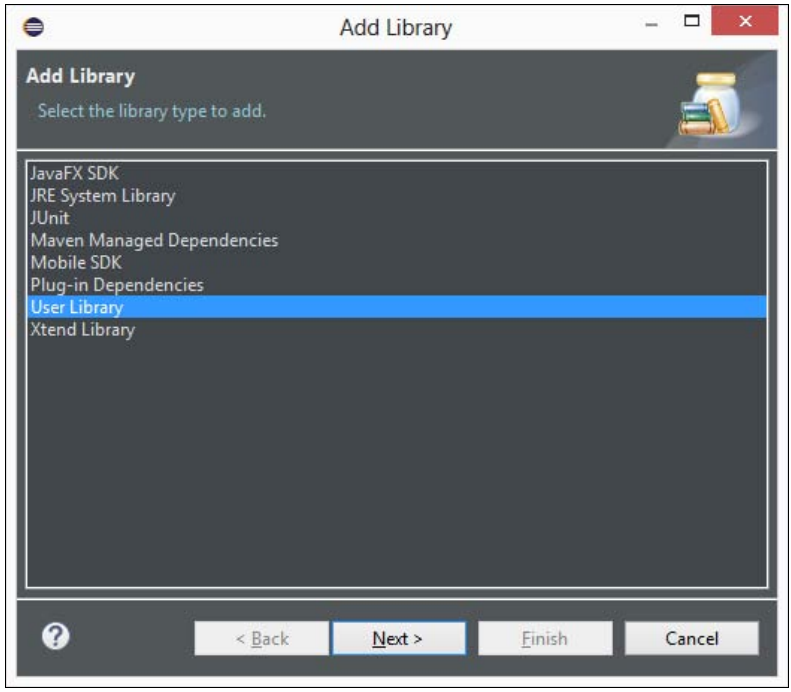


Рис. 1.10. Окно **Add Library**

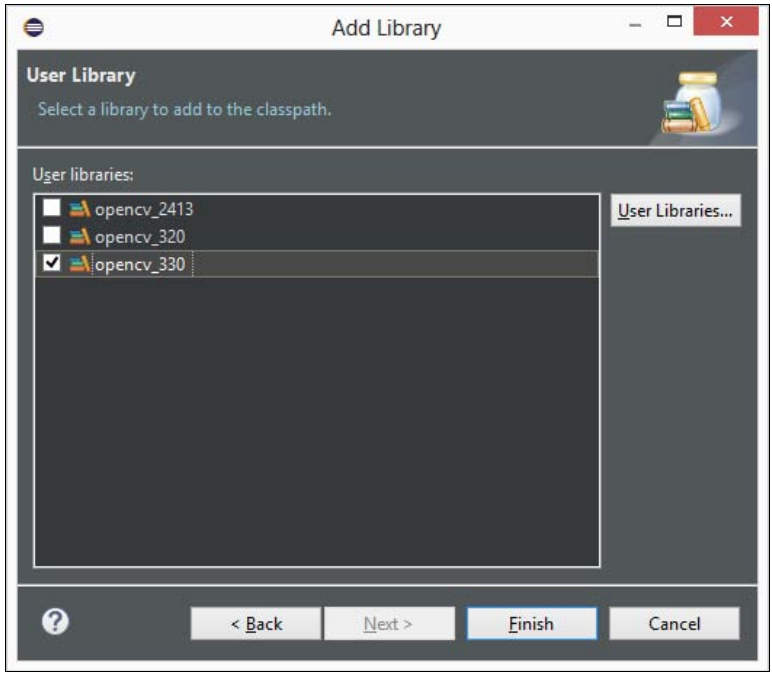


Рис. 1.11. Выбор нужной версии библиотеки OpenCV

Для проверки правильности установки создадим класс с названием `HelloCV` и внутри метода `main()` выведем версию библиотеки OpenCV, а также полную информацию о сборке (листинг 1.1).

Листинг 1.1. Вывод версии OpenCV

```
package application;

import org.opencv.core.Core;

public class HelloCV {
   static {
      System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
   }

   public static void main(String[] args) {
      System.out.println(Core.VERSION);              // 3.3.0
      System.out.println(Core.VERSION_MAJOR);        // 3
      System.out.println(Core.VERSION_MINOR);        // 3
      System.out.println(Core.VERSION_REVISION);     // 0
      System.out.println(Core.NATIVE_LIBRARY_NAME); // opencv_java330
      System.out.println(Core.getBuildInformation());
   }
}
```

Для компиляции и запуска программы в меню **Run** выбираем пункт **Run**, или нажимаем комбинацию клавиш <Ctrl>+<F11>, или нажимаем кнопку с белым треугольником внутри зеленого круга на панели инструментов. Результат выполнения будет выведен в окне **Console**.

В OpenCV версии 2.4 получить информацию о версии можно так:

```
System.out.println(Core.VERSION);              // 2.4.13.0
System.out.println(Core.VERSION_EPOCH);        // 2
System.out.println(Core.VERSION_MAJOR);        // 4
System.out.println(Core.VERSION_MINOR);        // 13
System.out.println(Core.VERSION_REVISION);     // 0
```

Константа `VERSION_EPOCH` в OpenCV версии 3.3 отсутствует. Как видите, уже с первых строк кода видны различия в версиях. Причем версии несовместимы друг с другом, т. к. различий очень много.

> ***Примечание***
>
> Если при запуске кода из листинга 1.1 вы получили какую-либо ошибку, или в консоли не отобразилась версия OpenCV, то, скорее всего, на компьютере не хватает библиотек динамической компоновки. Сначала нужно посмотреть на название архива с OpenCV. В версии 3.3.0 архив называется *opencv-3.3.0-vc14.exe*. Фрагмент *vc14* говорит, что библиотеки были скомпилированы в Visual Studio 2015. Следовательно, нужно установить

либо Visual Studio (можно бесплатную версию Community), либо бесплатный пакет Visual C++ Redistributable for Visual Studio 2015, который содержит все необходимые библиотеки. После установки не забудьте перезагрузить компьютер.

1.3. Вспомогательные классы

Прежде чем мы приступим к изучению основных классов, необходимо рассмотреть несколько вспомогательных классов, с помощью которых указываются различные параметры, — например: координаты точки, размеры области, положение области в пространстве и т. д.

1.3.1. Класс *Point*: точка

Класс `Point` описывает координаты точки в двумерном пространстве. Инструкция импорта:

```
import org.opencv.core.Point;
```

Конструкторы класса:

```
Point()
Point(double x, double y)
Point(double[] vals)
```

Пример использования первого конструктора:

```
Point p = new Point();
System.out.println(p);                // {0.0, 0.0}
System.out.println(p.x + " " + p.y); // 0.0 0.0
p.x = 10.0;
p.y = 5.0;
System.out.println(p.x + " " + p.y); // 10.0 5.0
```

Пример использования второго конструктора:

```
Point p = new Point(10.0, 5.0);
System.out.println(p);                // {10.0, 5.0}
System.out.println(p.x + " " + p.y); // 10.0 5.0
```

Пример использования третьего конструктора:

```
Point p = new Point(new double[] {10.0, 5.0});
System.out.println(p);                // {10.0, 5.0}
System.out.println(p.x + " " + p.y); // 10.0 5.0
```

Перечислим основные методы класса `Point`:

- `set()` — задает новое значение. Формат метода:

  ```
  public void set(double[] vals)
  ```

  Пример:

  ```
  Point p = new Point(0.0, 0.0);
  System.out.println(p.x + " " + p.y); // 0.0 0.0
  ```

```
p.set(new double[] {10.0, 5.0});
System.out.println(p.x + " " + p.y); // 10.0 5.0
p.x = 20.0;
p.y = 3.0;
System.out.println(p.x + " " + p.y); // 20.0 3.0
```

- `equals()` — выполняет сравнение двух точек. Формат метода:

```
public boolean equals(Object obj)
```

Пример:

```
Point p = new Point(0.0, 0.0);
Point p2 = new Point(0.0, 0.0);
Point p3 = new Point(5.0, 3.0);
System.out.println(p.equals(p2)); // true
System.out.println(p.equals(p3)); // false
```

- `clone()` — создает копию объекта. Формат метода:

```
public Point clone()
```

Пример:

```
Point p = new Point(10.0, 5.0);
Point p2 = p;
p2.x = 30.0;
System.out.println(p.x + " " + p.y);   // 30.0 5.0
Point p3 = p.clone();
p3.x = 85.0;
System.out.println(p.x + " " + p.y);   // 30.0 5.0
System.out.println(p3.x + " " + p3.y); // 85.0 5.0
```

- `inside()` — возвращает `true`, если точка входит в указанную прямоугольную область. Формат метода:

```
import org.opencv.core.Rect;
public boolean inside(Rect r)
```

Пример:

```
Point p = new Point(9.0, 5.0);
Rect rect = new Rect(0, 0, 10, 10);
Rect rect2 = new Rect(10, 5, 10, 10);
System.out.println(p.inside(rect));  // true
System.out.println(p.inside(rect2)); // false
```

Прочие методы:

```
public double dot(Point p)
public String toString()
public int hashCode()
```

1.3.2. Класс *Point3*: 3D-точка

Класс `Point3` описывает координаты точки в трехмерном пространстве. Инструкция импорта:

```
import org.opencv.core.Point3;
```

Конструкторы класса:

```
Point3()
Point3(double x, double y, double z)
Point3(double[] vals)
Point3(Point p)
```

Пример использования первого конструктора:

```
Point3 p = new Point3();
System.out.println(p);                          // {0.0, 0.0, 0.0}
System.out.println(p.x + " " + p.y + " " + p.z); // 0.0 0.0 0.0
p.x = 20.0;
p.y = 3.0;
p.z = 12.0;
System.out.println(p.x + " " + p.y + " " + p.z); // 20.0 3.0 12.0
```

Пример использования второго конструктора:

```
Point3 p = new Point3(10.0, 20.0, 30.0);
System.out.println(p.x + " " + p.y + " " + p.z); // 10.0 20.0 30.0
```

Пример использования третьего конструктора:

```
Point3 p = new Point3(new double[] {10.0, 20.0, 30.0});
System.out.println(p.x + " " + p.y + " " + p.z); // 10.0 20.0 30.0
```

Пример использования четвертого конструктора:

```
Point3 p = new Point3(new Point(10.0, 20.0));
System.out.println(p.x + " " + p.y + " " + p.z); // 10.0 20.0 0.0
```

Перечислим основные методы класса `Point3`:

- `set()` — задает новое значение. Формат метода:

  ```
  public void set(double[] vals)
  ```

  Пример:

  ```
  Point3 p = new Point3();
  p.set(new double[] {10.0, 20.0, 30.0});
  System.out.println(p.x + " " + p.y + " " + p.z);
  // 10.0 20.0 30.0
  ```

- `equals()` — выполняет сравнение двух точек. Формат метода:

  ```
  public boolean equals(Object obj)
  ```

  Пример:

  ```
  Point3 p = new Point3(10.0, 20.0, 30.0);
  Point3 p2 = new Point3(10.0, 20.0, 30.0);
  ```

```
Point3 p3 = new Point3(10.0, 20.0, 0.0);
System.out.println(p.equals(p2)); // true
System.out.println(p.equals(p3)); // false
```

- `clone()` — создает копию. Формат метода:

```
public Point3 clone()
```

Пример:

```
Point3 p = new Point3(10.0, 20.0, 30.0);
Point3 p2 = p;
p2.x = 0.0;
System.out.println(p.x + " " + p.y + " " + p.z);
// 0.0 20.0 30.0
Point3 p3 = p.clone();
p3.x = 85.0;
System.out.println(p.x + " " + p.y + " " + p.z);
// 0.0 20.0 30.0
System.out.println(p3.x + " " + p3.y + " " + p3.z);
// 85.0 20.0 30.0
```

Прочие методы:

```
public Point3 cross(Point3 p)
public double dot(Point3 p)
public int hashCode()
public String toString()
```

1.3.3. Класс *Size*: размеры прямоугольной области

Класс `Size` описывает размеры прямоугольной области. Инструкция импорта:

```
import org.opencv.core.Size;
```

Конструкторы класса:

```
Size()
Size(double width, double height)
Size(double[] vals)
Size(Point p)
```

Пример использования первого конструктора:

```
Size s = new Size();
System.out.println(s); // 0x0
s.width = 100.0;
s.height = 50.0;
System.out.println(s.width + " " + s.height); // 100.0 50.0
```

Пример использования второго конструктора:

```
Size s = new Size(100.0, 50.0);
System.out.println(s.width + " " + s.height); // 100.0 50.0
```

Пример использования третьего конструктора:

```
Size s = new Size(new double[] {100.0, 50.0});
System.out.println(s.width + " " + s.height); // 100.0 50.0
```

Пример использования четвертого конструктора:

```
Size s = new Size(new Point(100.0, 50.0));
System.out.println(s.width + " " + s.height); // 100.0 50.0
```

Перечислим основные методы класса `Size`:

- `set()` — задает новое значение. Формат метода:

  ```
  public void set(double[] vals)
  ```

  Пример:

  ```
  Size s = new Size(100.0, 50.0);
  s.set(new double[] {10.0, 20.0});
  System.out.println(s.width + " " + s.height); // 10.0 20.0
  ```

- `area()` — возвращает площадь. Формат метода:

  ```
  public double area()
  ```

  Пример:

  ```
  Size s = new Size(100.0, 50.0);
  System.out.println(s.area());           // 5000.0
  System.out.println(s.width * s.height); // 5000.0
  ```

- `empty()` — возвращает значение `true`, если ширина или высота меньше или равна нулю. Формат метода:

  ```
  public boolean empty() // Только в версии 3.3
  ```

  Пример:

  ```
  Size s = new Size(100.0, 50.0);
  System.out.println(s.empty());          // false
  Size s2 = new Size(100.0, 0.0);
  System.out.println(s2.empty());         // true
  ```

Прочие методы:

```
public Size clone()
public boolean equals(Object obj)
public int hashCode()
public String toString()
```

1.3.4. Класс *Rect*: прямоугольная область

Класс `Rect` описывает координаты и размеры прямоугольной области. Инструкция импорта:

```
import org.opencv.core.Rect;
```

Конструкторы класса:

```
Rect()
Rect(int x, int y, int width, int height)
Rect(double[] vals)
Rect(Point p1, Point p2)
Rect(Point p, Size s)
```

Пример использования первого конструктора:

```
Rect rect = new Rect();
System.out.println(rect); // {0, 0, 0x0}
rect.x = 0;
rect.y = 10;
rect.width = 100;
rect.height = 50;
System.out.println(rect.x + " " + rect.y);          // 0 10
System.out.println(rect.width + " " + rect.height); // 100 50
```

Пример использования второго конструктора:

```
Rect rect = new Rect(0, 0, 100, 50);
System.out.println(rect.x + " " + rect.y);          // 0 0
System.out.println(rect.width + " " + rect.height); // 100 50
```

Пример использования третьего конструктора:

```
Rect rect = new Rect(new double[] {0.0, 0.0, 100.0, 50.0});
System.out.println(rect.x + " " + rect.y);          // 0 0
System.out.println(rect.width + " " + rect.height); // 100 50
```

Пример использования четвертого конструктора:

```
Rect rect = new Rect(new Point(0.0, 0.0), new Point(100.0, 50.0));
System.out.println(rect.x + " " + rect.y);          // 0 0
System.out.println(rect.width + " " + rect.height); // 100 50
```

Пример использования пятого конструктора:

```
Rect rect = new Rect(new Point(0.0, 0.0), new Size(100.0, 50.0));
System.out.println(rect.x + " " + rect.y);          // 0 0
System.out.println(rect.width + " " + rect.height); // 100 50
```

Перечислим основные методы класса `Rect`:

- `set()` — задает новое значение. Формат метода:

  ```
  public void set(double[] vals)
  ```

  Пример:

  ```
  Rect rect = new Rect(10, 5, 100, 50);
  rect.set(new double[] {0.0, 0.0, 100.0, 50.0});
  System.out.println(rect.x + " " + rect.y);          // 0 0
  System.out.println(rect.width + " " + rect.height); // 100 50
  ```

- `size()` — возвращает размеры области. Формат метода:

```
public Size size()
```

Пример:

```
Rect rect = new Rect(10, 5, 100, 50);
Size s = rect.size();
System.out.println(s);          // 100x50
```

- `tl()` — возвращает координаты левого верхнего угла прямоугольной области. Формат метода:

```
public Point tl()
```

- `br()` — возвращает координаты правого нижнего угла прямоугольной области. Формат метода:

```
public Point br()
```

Пример:

```
Rect rect = new Rect(10, 5, 100, 50);
Point topLeft = rect.tl();
Point bottomRight = rect.br();
System.out.println(topLeft);          // {10.0, 5.0}
System.out.println(bottomRight);      // {110.0, 55.0}
```

- `area()` — возвращает площадь прямоугольника. Формат метода:

```
public double area()
```

Пример:

```
Rect rect = new Rect(10, 5, 100, 50);
System.out.println(rect.area());          // 5000.0
```

- `contains()` — возвращает `true`, если указанная точка входит в прямоугольную область. Формат метода:

```
public boolean contains(Point p)
```

Пример:

```
Rect rect = new Rect(10, 5, 100, 50);
System.out.println(rect.contains(new Point(10.0, 5.0))); // true
System.out.println(rect.contains(new Point(9.0, 5.0)));  // false
```

- `empty()` — возвращает значение `true`, если ширина или высота меньше или равна нулю. Формат метода:

```
public boolean empty() // Только в версии 3.3
```

Пример:

```
Rect rect = new Rect(10, 5, 100, 50);
System.out.println(rect.empty());     // false
Rect rect2 = new Rect(10, 5, 100, 0);
System.out.println(rect2.empty());    // true
```

Прочие методы:

```
public Rect clone()
public boolean equals(Object obj)
public int hashCode()
public String toString()
```

В версии 3.3 доступен также класс `Rect2d`, который имеет точно такие же конструкторы и методы, как и класс `Rect`. Различие состоит в типе данных полей класса: в классе `Rect` поля `x`, `y`, `width` и `height` имеют тип `int`, а в классе `Rect2d` — тип `double`. Инструкция импорта:

```
import org.opencv.core.Rect2d;
```

Пример:

```
Rect2d rect = new Rect2d(10, 5, 100, 50);
System.out.println(rect);        // {10.0, 5.0, 100.0x50.0}
rect.set(new double[] {0.0, 0.0, 100.0, 50.0});
System.out.println(rect.size()); // 100x50
System.out.println(rect.tl());   // {0.0, 0.0}
```

1.3.5. Класс *RotatedRect*: повернутая прямоугольная область

Класс `RotatedRect` описывает прямоугольную область, которая повернута на некоторый угол. Инструкция импорта:

```
import org.opencv.core.RotatedRect;
```

Конструкторы класса:

```
RotatedRect()
RotatedRect(Point center, Size size, double angle)
RotatedRect(double[] vals)
```

Параметр `center` задает координаты центра прямоугольника, параметр `size` — размеры прямоугольника, а параметр `angle` — угол, на который повернут прямоугольник.

Пример использования первого конструктора:

```
RotatedRect rect = new RotatedRect();
System.out.println(rect); // { {0.0, 0.0} 0x0 * 0.0 }
rect.center = new Point(50.0, 50.0);
rect.size = new Size(100.0, 100.0);
rect.angle = 45.0;
System.out.println(rect); // { {50.0, 50.0} 100x100 * 45.0 }
```

Пример использования второго конструктора:

```
RotatedRect rect = new RotatedRect(new Point(50.0, 50.0),
                       new Size(100.0, 100.0), 45.0);
System.out.println(rect); // { {50.0, 50.0} 100x100 * 45.0 }
```

Пример использования третьего конструктора:

```
RotatedRect rect = new RotatedRect(new double[] {50.0, 50.0,
                                                 100.0, 100.0, 45.0});
System.out.println(rect);        // { {50.0, 50.0} 100x100 * 45.0 }
System.out.println(rect.center); // {50.0, 50.0}
System.out.println(rect.size);   // 100x100
System.out.println(rect.angle);  // 45.0
```

Перечислим основные методы класса `RotatedRect`:

- `set()` — задает новое значение. Формат метода:

  ```
  public void set(double[] vals)
  ```

  Пример:

  ```
  RotatedRect rect = new RotatedRect();
  rect.set(new double[] {50.0, 50.0, 100.0, 100.0, 45.0});
  System.out.println(rect);  // { {50.0, 50.0} 100x100 * 45.0 }
  ```

- `boundingRect()` — возвращает прямоугольную область, в которую вписан повернутый прямоугольник. Формат метода:

  ```
  public Rect boundingRect()
  ```

  Пример:

  ```
  RotatedRect rect = new RotatedRect(new Point(50.0, 50.0),
                         new Size(100.0, 100.0), 45.0);
  System.out.println(rect.boundingRect());
  // {-21, -21, 143x143}
  ```

- `points()` — записывает в массив координаты вершин прямоугольника. Формат метода:

  ```
  public void points(Point pt[])
  ```

  Пример:

  ```
  RotatedRect rect = new RotatedRect(new Point(50.0, 50.0),
                         new Size(100.0, 100.0), 45.0);
  Point[] pt = new Point[4];
  rect.points(pt);
  for (int i = 0; i < pt.length; i++) {
     System.out.println(pt[i]);
  }
  /*
  {-20.710678118654748, 50.00000000000001}
  {50.0, -20.710678118654748}
  {120.71067811865476, 49.99999999999999}
  {50.0, 120.71067811865476}*/
  ```

Прочие методы:

```
public RotatedRect clone()
public boolean equals(Object obj)
public int hashCode()
public String toString()
```

1.3.6. Класс *Scalar*: объект из четырех элементов

Класс `Scalar` описывает объект из четырех элементов. Используется в основном для указания скалярного значения, а также для описания значений компонентов цвета. Инструкция импорта:

```
import org.opencv.core.Scalar;
```

Конструкторы класса:

```
Scalar(double v0)
Scalar(double v0, double v1)
Scalar(double v0, double v1, double v2)
Scalar(double v0, double v1, double v2, double v3)
Scalar(double[] vals)
```

Пример:

```
Scalar s = new Scalar(1.0);
System.out.println(s); // [1.0, 0.0, 0.0, 0.0]
s = new Scalar(1.0, 2.0);
System.out.println(s); // [1.0, 2.0, 0.0, 0.0]
s = new Scalar(1.0, 2.0, 3.0);
System.out.println(s); // [1.0, 2.0, 3.0, 0.0]
s = new Scalar(1.0, 2.0, 3.0, 4.0);
System.out.println(s); // [1.0, 2.0, 3.0, 4.0]
s = new Scalar(new double[] {1.0, 2.0, 3.0, 4.0});
System.out.println(s); // [1.0, 2.0, 3.0, 4.0]
System.out.println(s.val[0]); // 1.0
System.out.println(s.val[1]); // 2.0
System.out.println(s.val[2]); // 3.0
System.out.println(s.val[3]); // 4.0
s.val[0] = 10.0;
s.val[1] = 20.0;
s.val[2] = 30.0;
s.val[3] = 40.0;
System.out.println(s); // [10.0, 20.0, 30.0, 40.0]
```

Для создания объекта с одинаковыми значениями всех компонентов можно воспользоваться статическим методом `all()`. Формат метода:

```
public static Scalar all(double v)
```

Пример:

```
Scalar s = Scalar.all(1.0);
System.out.println(s); // [1.0, 1.0, 1.0, 1.0]
```

Перечислим основные методы класса `Scalar`:

- `set()` — задает новое значение. Формат метода:

```
public void set(double[] vals)
```

Пример:

```
Scalar s = new Scalar(0.0);
s.set(new double[] {1.0, 2.0, 3.0, 4.0});
System.out.println(s); // [1.0, 2.0, 3.0, 4.0]
s.set(new double[] {10.0});
System.out.println(s); // [10.0, 0.0, 0.0, 0.0]
```

- `isReal()` — возвращает значение `true`, если второй, третий и четвертый компоненты равны `0.0`. Формат метода:

```
public boolean isReal()
```

Пример:

```
Scalar s = Scalar.all(1.0);
System.out.println(s.isReal()); // false
s = Scalar.all(0.0);
System.out.println(s.isReal()); // true
s = new Scalar(10, 0, 0, 0);
System.out.println(s.isReal()); // true
```

- `conj()` — возвращает значение `(v0, -v1, -v2, -v3)`. Формат метода:

```
public Scalar conj()
```

Пример:

```
Scalar s = new Scalar(1.0, 2.0, 3.0, 4.0);
System.out.println(s.conj()); // [1.0, -2.0, -3.0, -4.0]
```

- `mul()` — перемножает значения компонентов (которые расположены на одинаковых позициях) двух объектов и возвращает новый объект. Параметр `scale` задает дополнительный коэффициент масштаба (в первом формате параметр равен `1`). Форматы метода:

```
public Scalar mul(Scalar it)
public Scalar mul(Scalar it, double scale)
```

Пример:

```
Scalar s = new Scalar(1.0, 2.0, 3.0, 4.0);
System.out.println(s.mul(Scalar.all(2.0)));
// [2.0, 4.0, 6.0, 8.0]
s = new Scalar(1.0, 2.0, 3.0, 4.0);
System.out.println(s.mul(Scalar.all(2.0), 2));
// [4.0, 8.0, 12.0, 16.0]
```

Прочие методы:

```
public Scalar clone()
public boolean equals(Object obj)
```

```
public int hashCode()
public String toString()
```

1.3.7. Класс *Range*: диапазон

Класс `Range` описывает диапазон значений. Инструкция импорта:

```
import org.opencv.core.Range;
```

Конструкторы класса:

```
Range()
Range(int start, int end)
Range(double[] vals)
```

Параметр `start` задает начало диапазона (включая элемент с этим значением), а параметр `end` — конец диапазона (не включая элемент с этим значением).

Пример использования первого конструктора:

```
Range r = new Range();
r.start = 1;
r.end = 51;
System.out.println(r);       // [1, 51)
```

Пример использования второго конструктора:

```
Range r = new Range(1, 51);
System.out.println(r);       // [1, 51)
System.out.println(r.start); // 1
System.out.println(r.end);   // 51
```

Пример использования третьего конструктора:

```
Range r = new Range(new double[] {1, 51});
System.out.println(r);        // [1, 51)
```

С помощью статического метода `all()` можно создать диапазон всех возможных значений. Формат метода:

```
public static Range all()
```

Пример:

```
Range r = Range.all();
System.out.println(r);        // [-2147483648, 2147483647)
System.out.println(r.size()); // -1
```

Перечислим основные методы класса `Range`:

- `set()` — задает новые значения. Формат метода:

  ```
  public void set(double[] vals)
  ```

  Пример:

  ```
  Range r = new Range();
  r.set(new double[] {1, 51});
  System.out.println(r);        // [1, 51)
  ```

- `size()` — возвращает количество элементов, входящих в диапазон. Формат метода:

  ```
  public int size()
  ```

  Пример:

  ```
  Range r = new Range(1, 51);
  System.out.println(r.size()); // 50
  ```

- `empty()` — возвращает значение `true`, если диапазон не содержит элементов. Формат метода:

  ```
  public boolean empty()
  ```

  Пример:

  ```
  Range r = new Range();
  System.out.println(r);         // [0, 0)
  System.out.println(r.size());  // 0
  System.out.println(r.empty()); // true
  ```

- `shift()` — сдвигает границы диапазона на указанное значение. Формат метода:

  ```
  public Range shift(int delta)
  ```

  Пример:

  ```
  Range r = new Range(1, 51);
  System.out.println(r.shift(10));         // [11, 61)
  ```

- `intersection()` — возвращает пересечение двух диапазонов. Формат метода:

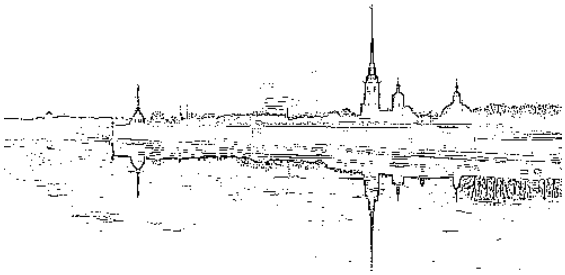
  ```
  public Range intersection(Range r1)
  ```

  Пример:

  ```
  Range r = new Range(1, 51);
  Range r2 = new Range(30, 81);
  System.out.println(r.intersection(r2));  // [30, 51)
  ```

Прочие методы:

```
public Range clone()
public boolean equals(Object obj)
public int hashCode()
public String toString()
```



ГЛАВА 2

Матрицы

Базовыми понятиями в OpenCV являются: *скаляр*, *вектор* и *матрица*. Скалярное значение задается с помощью класса `Scalar` (см. *разд. 1.3.6*). Матрицу реализует класс `Mat`, который мы начнем рассматривать в этой главе. Вектор в OpenCV в Java реализуется с помощью одномерной матрицы. Для удобства создания векторов предназначены классы, наследующие класс `Mat`, — например, `MatOfInt` (вектор с целочисленными значениями), `MatOfPoint` (вектор с координатами точек в двумерном пространстве) и т. д.

2.1. Класс *Mat*: матрица

Класс `Mat` описывает матрицу, которая, например, служит для хранения изображений. Инструкция импорта:

```
import org.opencv.core.Mat;
```

2.1.1. Создание матрицы

Создать матрицу позволяют следующие конструкторы:

```
Mat()
Mat(int rows, int cols, int type)
Mat(int rows, int cols, int type, Scalar s)
Mat(Size size, int type)
Mat(Size size, int type, Scalar s)
Mat(long addr)
```

Параметр `rows` задает количество строк (высоту для изображения), а параметр `cols` — количество столбцов (ширину для изображения). Задать размеры можно также с помощью параметра `size`. В параметре `type` указывается тип элементов матрицы. С помощью параметра `s` можно заполнить матрицу одинаковым значением. Если значение не присвоить, то элементы будут иметь произвольные значения.

Пример создания пустой матрицы и вывода информации о ней:

```
Mat m = new Mat();
System.out.println(m.dump());   // []
System.out.println(m.cols());   // 0
System.out.println(m.width());  // 0
System.out.println(m.rows());   // 0
System.out.println(m.height()); // 0
```

Пример создания матрицы типа `CV_8UC1` из трех строк и двух столбцов (каждый элемент такой матрицы будет иметь один канал, который может содержать 8-битное целое число без знака в диапазоне от `0` до `255`):

```
// import org.opencv.core.CvType;
Mat m = new Mat(3, 2, CvType.CV_8UC1);
System.out.println(m.dump());
/*
[  0,   0;
   0,   0;
   0,   0]
*/
System.out.println(m.cols());   // 2
System.out.println(m.width());  // 2
System.out.println(m.rows());   // 3
System.out.println(m.height()); // 3
```

Матрица из предыдущего примера будет заполнена произвольными значениями (не обязательно нулями). Чтобы присвоить элементам матрицы начальное значение, следует использовать параметр `s`:

```
Mat m = new Mat(3, 2, CvType.CV_8UC1, new Scalar(1.0));
System.out.println(m.dump());
/*
[  1,   1;
   1,   1;
   1,   1]
*/
```

При создании изображения удобнее задавать не количество строк и столбцов, а указать размеры изображения с помощью объекта класса `Size`.

Пример создания матрицы, каждый элемент которой содержит три канала:

```
Mat m = new Mat(new Size(2, 3), CvType.CV_8UC3);
System.out.println(m.dump());
/*
[  0,   0,   0,   0,   0,   0;
   0,   0, 211, 190, 218, 109;
  46,  22,   0, 128,   0, 109]
*/
```

```
System.out.println(m.cols());   // 2
System.out.println(m.width());  // 2
System.out.println(m.rows());   // 3
System.out.println(m.height()); // 3
```

Чтобы задать начальные значения, следует использовать параметр `s`:

```
Mat m = new Mat(new Size(2, 3), CvType.CV_8UC3, new Scalar(1, 2, 3));
System.out.println(m.dump());
/*
[  1,   2,   3,   1,   2,   3;
   1,   2,   3,   1,   2,   3;
   1,   2,   3,   1,   2,   3]
*/
```

Создать матрицу, заполненную нулями, позволяет статический метод `zeros()`. Форматы метода:

```
public static Mat zeros(int rows, int cols, int type)
public static Mat zeros(Size size, int type)
```

Пример:

```
Mat m = Mat.zeros(3, 2, CvType.CV_8UC1);
System.out.println(m.dump());
/*
[  0,   0;
   0,   0;
   0,   0]
*/
Mat m2 = Mat.zeros(new Size(2, 3), CvType.CV_8UC3);
System.out.println(m2.dump());
/*
[  0,   0,   0,   0,   0,   0;
   0,   0,   0,   0,   0,   0;
   0,   0,   0,   0,   0,   0]
*/
```

Создать матрицу, заполненную единицами, позволяет статический метод `ones()`. Форматы метода:

```
public static Mat ones(int rows, int cols, int type)
public static Mat ones(Size size, int type)
```

Пример:

```
Mat m = Mat.ones(3, 2, CvType.CV_8UC1);
System.out.println(m.dump());
/*
[  1,   1;
   1,   1;
   1,   1]
*/
```

```
Mat m2 = Mat.ones(new Size(2, 3), CvType.CV_8UC3);
System.out.println(m2.dump());
/*
[  1,   0,   0,   1,   0,   0;
   1,   0,   0,   1,   0,   0;
   1,   0,   0,   1,   0,   0]
*/
```

Создать матрицу, заполненную единицами по диагонали, позволяет статический метод `eye()`. Форматы метода:

```
public static Mat eye(int rows, int cols, int type)
public static Mat eye(Size size, int type)
```

Пример:

```
Mat m = Mat.eye(3, 3, CvType.CV_8UC1);
System.out.println(m.dump());
/*
[  1,   0,   0;
   0,   1,   0;
   0,   0,   1]
*/
Mat m2 = Mat.eye(new Size(3, 3), CvType.CV_8UC3);
System.out.println(m2.dump());
/*
[  1,   0,   0,   0,   0,   0,   0,   0,   0;
   0,   0,   0,   1,   0,   0,   0,   0,   0;
   0,   0,   0,   0,   0,   0,   1,   0,   0]
*/
```

Если нужно заполнить уже существующую матрицу единицами по диагонали, то можно воспользоваться статическим методом `setIdentity()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void setIdentity(Mat m)
public static void setIdentity(Mat m, Scalar s)
```

Если параметр `s` не указан, то матрица по диагонали заполняется единицами, в противном случае — значением параметра `s`. Элементы вне диагонали заполняются нулями. Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
Core.setIdentity(m);
System.out.println(m.dump());
/*
[  1,   0,   0;
   0,   1,   0;
   0,   0,   1]
*/
```

```
Core.setIdentity(m, new Scalar(5));
System.out.println(m.dump());
/*
[  5,   0,   0;
   0,   5,   0;
   0,   0,   5]
*/
```

Создать матрицу и заполнить ее значениями по диагонали из другой одномерной матрицы позволяет статический метод `diag()`. Формат метода:

```
public static Mat diag(Mat d)
```

Пример:

```
Mat m = new Mat(1, 3, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
System.out.println(m.dump()); // [  1,   2,   3]
Mat m2 = Mat.diag(m);
System.out.println(m2.dump());
/*
[  1,   0,   0;
   0,   2,   0;
   0,   0,   3]
*/
```

Изменить структуру матрицы после ее создания позволяет метод `create()`. Форматы метода:

```
public void create(int rows, int cols, int type)
public void create(Size size, int type)
```

Пример:

```
Mat m = new Mat();
m.create(2, 3, CvType.CV_8UC1);
m.setTo(new Scalar(1));
System.out.println(m.dump());
/*
[  1,   1,   1;
   1,   1,   1]
*/
m.create(new Size(2, 2), CvType.CV_8UC1);
m.setTo(new Scalar(5));
System.out.println(m.dump());
/*
```

```
[  5,   5;
   5,   5]
*/
```

Для вывода значений матрицы в этих примерах мы воспользовались методом `dump()`. Чтобы получить информацию о структуре матрицы в виде строки, нужно использовать метод `toString()`. Форматы методов:

```
public String dump()
public String toString()
```

Пример:

```
Mat m = new Mat(1, 3, CvType.CV_8UC1, new Scalar(5));
System.out.println(m.dump());     // [  5,   5,   5]
System.out.println(m.toString());
// Mat [ 1*3*CV_8UC1, isCont=true, isSubmat=false,
// nativeObj=0xb11400, dataAddr=0xb14580 ]
```

2.1.2. Размеры матрицы

Получить информацию о размерах матрицы позволяют следующие методы из класса `Mat`:

- `rows()` — возвращает количество строк (высоту для изображения). Формат метода:

  ```
  public int rows()
  ```

- `cols()` — возвращает количество столбцов (ширину для изображения). Формат метода:

  ```
  public int cols()
  ```

- `width()` — возвращает ширину для изображения. Формат метода:

  ```
  public int width()
  ```

- `height()` — возвращает высоту для изображения. Формат метода:

  ```
  public int height()
  ```

- `size()` — возвращает размеры изображения (`Size(cols, rows)`). Формат метода:

  ```
  public Size size()
  ```

  Пример:

  ```
  Mat m = new Mat(3, 2, CvType.CV_8UC1);
  System.out.println(m.cols());   // 2
  System.out.println(m.width());  // 2
  System.out.println(m.rows());   // 3
  System.out.println(m.height()); // 3
  System.out.println(m.size());   // 2x3
  ```

- `empty()` — возвращает значение `true`, если матрица не содержит элементов. Формат метода:

```
public boolean empty()
```

Пример:

```
Mat m = new Mat(3, 2, CvType.CV_8UC1);
System.out.println(m.empty());  // false
m = new Mat();
System.out.println(m.empty());  // true
```

- `total()` — возвращает количество элементов матрицы. Формат метода:

```
public long total()
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
System.out.println(m.total()); // 9
```

- `elemSize()` — возвращает размер элемента матрицы в байтах. Формат метода:

```
public long elemSize()
```

- `elemSize1()` — возвращает размер канала в байтах. Формат метода:

```
public long elemSize1()
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC3);
System.out.println(m.elemSize());   // 3
System.out.println(m.elemSize1());  // 1
Mat m2 = new Mat(3, 3, CvType.CV_32FC3);
System.out.println(m2.elemSize());  // 12
System.out.println(m2.elemSize1()); // 4
```

2.1.3. Тип элементов

При создании матрицы в параметре `type` нужно указать тип элементов. В качестве значения указываются следующие константы из класса `CvType`:

```
import org.opencv.core.CvType;
public static final int CV_8SC1
public static final int CV_8SC2
public static final int CV_8SC3
public static final int CV_8SC4
public static final int CV_8UC1
public static final int CV_8UC2
public static final int CV_8UC3
public static final int CV_8UC4
public static final int CV_16SC1
public static final int CV_16SC2
public static final int CV_16SC3
```

```
public static final int CV_16SC4
public static final int CV_16UC1
public static final int CV_16UC2
public static final int CV_16UC3
public static final int CV_16UC4
public static final int CV_32SC1
public static final int CV_32SC2
public static final int CV_32SC3
public static final int CV_32SC4
public static final int CV_32FC1
public static final int CV_32FC2
public static final int CV_32FC3
public static final int CV_32FC4
public static final int CV_64FC1
public static final int CV_64FC2
public static final int CV_64FC3
public static final int CV_64FC4
```

Цифра после символа подчеркивания означает количество битов, выделяемое на один канал. Далее идет буква `S` (целое число со знаком), `U` (целое число без знака) или `F` (вещественное число), которая задает тип данных. После буквы `C` указывается количество каналов (от 1 до 4). Выведем значения констант и диапазоны значений:

```
// 8 бит (-128 ... 127)
System.out.println(CvType.CV_8SC1);  // 1
System.out.println(CvType.CV_8SC2);  // 9
System.out.println(CvType.CV_8SC3);  // 17
System.out.println(CvType.CV_8SC4);  // 25
// 8 бит (0 ... 255)
System.out.println(CvType.CV_8UC1);  // 0
System.out.println(CvType.CV_8UC2);  // 8
System.out.println(CvType.CV_8UC3);  // 16
System.out.println(CvType.CV_8UC4);  // 24
// 16 бит (-32768 ... 32767)
System.out.println(CvType.CV_16SC1); // 3
System.out.println(CvType.CV_16SC2); // 11
System.out.println(CvType.CV_16SC3); // 19
System.out.println(CvType.CV_16SC4); // 27
// 16 бит (0 ... 65535)
System.out.println(CvType.CV_16UC1); // 2
System.out.println(CvType.CV_16UC2); // 10
System.out.println(CvType.CV_16UC3); // 18
System.out.println(CvType.CV_16UC4); // 26
// 32 бита (-2 147 483 648 ... 2 147 483 647)
System.out.println(CvType.CV_32SC1); // 4
System.out.println(CvType.CV_32SC2); // 12
System.out.println(CvType.CV_32SC3); // 20
System.out.println(CvType.CV_32SC4); // 28
```

```
// 32 бита float
System.out.println(CvType.CV_32FC1); // 5
System.out.println(CvType.CV_32FC2); // 13
System.out.println(CvType.CV_32FC3); // 21
System.out.println(CvType.CV_32FC4); // 29
// 64 бита double
System.out.println(CvType.CV_64FC1); // 6
System.out.println(CvType.CV_64FC2); // 14
System.out.println(CvType.CV_64FC3); // 22
System.out.println(CvType.CV_64FC4); // 30
```

Для указания типа элементов матрицы можно также воспользоваться следующими статическими методами из класса `CvType`:

```
public static final int CV_8UC(int ch)
public static final int CV_8SC(int ch)
public static final int CV_16UC(int ch)
public static final int CV_16SC(int ch)
public static final int CV_32SC(int ch)
public static final int CV_32FC(int ch)
public static final int CV_64FC(int ch)
```

С помощью этих методов можно создать матрицу, элементы которых будут содержать более четырех каналов (от 1 до 511 каналов). Пример:

```
Mat m = new Mat(1, 2, CvType.CV_8UC(7));
System.out.println(m.channels());  // 7
Mat m2 = new Mat(1, 2, CvType.CV_8UC(511));
System.out.println(m2.channels()); // 511
```

Для указания типа с произвольной глубиной и количеством каналов можно воспользоваться статическим методом `makeType()` из класса `CvType`. Формат метода:

```
public static final int makeType(int depth, int channels)
```

Пример:

```
Mat m = new Mat(1, 2, CvType.makeType(CvType.CV_8U, 7));
System.out.println(m.channels());  // 7
System.out.println(m.depth());     // 0
```

Получить информацию о типе элементов матрицы, о количестве каналов и глубине позволяют следующие методы из класса `Mat`:

```
public int type()
public int channels()
public int depth()
```

Пример:

```
Mat m = new Mat(3, 2, CvType.CV_8UC1);
System.out.println(m.type());     // 0
System.out.println(m.channels()); // 1
System.out.println(m.depth());    // 0
```

```
m = new Mat(3, 2, CvType.CV_8SC2);
System.out.println(m.channels()); // 2
System.out.println(m.depth());    // 1
m = new Mat(3, 2, CvType.CV_16UC3);
System.out.println(m.channels()); // 3
System.out.println(m.depth());    // 2
m = new Mat(3, 2, CvType.CV_16SC3);
System.out.println(m.channels()); // 3
System.out.println(m.depth());    // 3
m = new Mat(3, 2, CvType.CV_32SC3);
System.out.println(m.channels()); // 3
System.out.println(m.depth());    // 4
m = new Mat(3, 2, CvType.CV_32FC4);
System.out.println(m.channels()); // 4
System.out.println(m.depth());    // 5
m = new Mat(3, 2, CvType.CV_64FC4);
System.out.println(m.channels()); // 4
System.out.println(m.depth());    // 6
```

Метод `depth()` возвращает значение следующих статических констант из класса `CvType`:

```
import org.opencv.core.CvType;
public static final int CV_8U
public static final int CV_8S
public static final int CV_16U
public static final int CV_16S
public static final int CV_32S
public static final int CV_32F
public static final int CV_64F
public static final int CV_USRTYPE1
```

Выведем их значения:

```
System.out.println(CvType.CV_8U);       // 0
System.out.println(CvType.CV_8S);       // 1
System.out.println(CvType.CV_16U);      // 2
System.out.println(CvType.CV_16S);      // 3
System.out.println(CvType.CV_32S);      // 4
System.out.println(CvType.CV_32F);      // 5
System.out.println(CvType.CV_64F);      // 6
System.out.println(CvType.CV_USRTYPE1); // 7
```

Получить информацию о типе матрицы в виде строки позволяет статический метод `typeToString()` из класса `CvType`. Формат метода:

```
public static final String typeToString(int type)
```

Пример:

```
System.out.println(CvType.typeToString(CvType.CV_8UC4));
// CV_8UC4
```

```
System.out.println(CvType.typeToString(CvType.CV_8UC(7)));
// CV_8UC(7)
```

С помощью метода `convertTo()` из класса `Mat` можно изменить тип матрицы. Форматы метода:

```
public void convertTo(Mat m, int rtype)
public void convertTo(Mat m, int rtype, double alpha)
public void convertTo(Mat m, int rtype, double alpha, double beta)
```

В первом параметре указываем ссылку на матрицу, в которую будут скопированы данные, а во втором параметре — новый тип. Значение параметра `alpha` будет умножено на значение элемента, а значение `beta` — прибавлено.

Пример преобразования типа `CV_8U` в `CV_32F` с пересчетом значений компонентов цвета:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(43, 0, 255));
System.out.println(m.dump()); // [43, 0, 255]
Mat m2 = new Mat();
m.convertTo(m2, CvType.CV_32F, 1.0 / 255);
System.out.println(m2.dump()); // [0.16862746, 0, 1]
```

Выполнить преобразование матрицы в тип `CV_8U` можно с помощью статического метода `convertScaleAbs()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void convertScaleAbs(Mat src, Mat dst)
public static void convertScaleAbs(Mat src, Mat dst,
                                   double alpha, double beta)
```

В первом параметре указываем ссылку на исходную матрицу, а во втором параметре — ссылку на матрицу, в которую будет записан результат операции. Значение параметра `alpha` будет умножено на значение элемента, а значение `beta` — прибавлено. После этого знак числа будет отброшен, т. е. отрицательное число станет положительным. Далее проверяется соответствие значения диапазону допустимых значений. Если значение больше `255`, то оно становится равным `255`. Пример:

```
Mat m = new Mat(1, 1, CvType.CV_16SC4, new Scalar(-43, 0, 255, 300));
System.out.println(m.dump()); // [-43, 0, 255, 300]
Mat m2 = new Mat();
Core.convertScaleAbs(m, m2);
System.out.println(m2.dump()); // [43, 0, 255, 255]
Mat m3 = new Mat();
m.convertTo(m3, CvType.CV_8U);
System.out.println(m3.dump()); // [0, 0, 255, 255]
```

Обратите внимание: при использовании метода `convertScaleAbs()` значение `-43` стало положительным (т. к. вычисляется абсолютное значение), а при использовании метода `convertTo()` стало равно `0` (т. к. отрицательное значение не входит в диапазон `0...255`).

2.1.4. Доступ к элементам

Изменить значение элемента матрицы позволяет метод `put()`. Форматы метода:

```
public int put(int row, int col, double... data)
public int put(int row, int col, byte[] data)    // CV_8U или CV_8S
public int put(int row, int col, short[] data)   // CV_16U или CV_16S
public int put(int row, int col, int[] data)     // CV_32S
public int put(int row, int col, float[] data)   // CV_32F
```

Получить значение позволяет метод `get()`. Форматы метода:

```
public double[] get(int row, int col)
public int get(int row, int col, byte[] data)    // CV_8U или CV_8S
public int get(int row, int col, short[] data)   // CV_16U или CV_16S
public int get(int row, int col, int[] data)     // CV_32S
public int get(int row, int col, float[] data)   // CV_32F
public int get(int row, int col, double[] data)  // CV_64F
```

Заполним матрицу значениями, а затем получим эти значения и выведем в консоль:

```
Mat m = new Mat(3, 2, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
double[] arr = null;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      arr = m.get(i, j);
      System.out.print(arr[0] + " ");
   }
} // 1.0 2.0 3.0 4.0 5.0 6.0
System.out.println(m.dump());
/*
[  1,   2;
   3,   4;
   5,   6]
*/
```

Если в первых двух параметрах метода `put()` указать нулевые значения, а в третьем — массив, то значения из массива будут скопированы в матрицу:

```
Mat m = new Mat(3, 2, CvType.CV_8UC1);
byte[] barr = {1, 2, 3, 4, 5, 6};
m.put(0, 0, barr);
System.out.println(m.dump());
/*
```

```
[  1,   2;
   3,   4;
   5,   6]
*/
```

Если в первых двух параметрах метода `get()` указать нулевые значения, а в третьем — передать ссылку на массив соответствующего размера, то значения из матрицы будут скопированы в массив. Метод вернет количество скопированных элементов. Пример:

```
// import java.util.Arrays;
Mat m = new Mat(3, 2, CvType.CV_8UC1);
m.put(0, 0, new byte[] {1, 2, 3, 4, 5, 6});
byte[] arr = new byte[m.channels() * m.cols() * m.rows()];
System.out.println(m.get(0, 0, arr));     // 6
System.out.println(Arrays.toString(arr)); // [1, 2, 3, 4, 5, 6]
```

Задать значение сразу всем элементам матрицы позволяет метод `setTo()`. Форматы метода:

```
public Mat setTo(Scalar s)
public Mat setTo(Scalar value, Mat mask)
public Mat setTo(Mat value)
public Mat setTo(Mat value, Mat mask)
```

Пример:

```
Mat m = new Mat(2, 3, CvType.CV_8UC1);
m.setTo(new Scalar(1));
System.out.println(m.dump());
/*
[  1,   1,   1;
   1,   1,   1]
*/
```

2.1.5. Получение диапазона значений

Получить *диапазон значений* (*субматрицу*) позволяют следующие методы:

- `row()` — возвращает матрицу, содержащую элементы указанной строки. Изменение значений в этой матрице затронет и значения исходной матрицы. Формат метода:

  ```
  public Mat row(int y)
  ```

  Пример:

  ```
  Mat m = new Mat(3, 2, CvType.CV_8UC1);
  double n = 1.0;
  for (int i = 0, r = m.rows(); i < r; i++) {
     for (int j = 0, c = m.cols(); j < c; j++) {
        m.put(i, j, n++);
     }
  }
  ```

```
Mat m2 = m.row(0);
System.out.println(m2.dump()); // [  1,   2]
m2.put(0, 0, 55);
System.out.println(m2.dump()); // [ 55,   2]
System.out.println(m.dump());
/*
[ 55,   2;
   3,   4;
   5,   6]
*/
```

- `rowRange()` — возвращает матрицу, содержащую элементы из диапазона строк. Изменение значений в этой матрице затронет и значения исходной матрицы. Форматы метода:

```
public Mat rowRange(int startrow, int endrow)
public Mat rowRange(Range r)
```

Пример:

```
Mat m = new Mat(3, 2, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
// Mat m2 = m.rowRange(0, 2);
Mat m2 = m.rowRange(new Range(0, 2));
System.out.println(m2.dump());
/*
[  1,   2;
   3,   4]
*/
m2.put(0, 0, 55);
System.out.println(m2.dump());
/*
[ 55,   2;
   3,   4]
*/
System.out.println(m.dump());
/*
[ 55,   2;
   3,   4;
   5,   6]
*/
```

- `col()` — возвращает матрицу, содержащую элементы указанного столбца. Изменение значений в этой матрице затронет и значения исходной матрицы. Формат метода:

```
public Mat col(int x)
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
Mat m2 = m.col(0);
System.out.println(m2.dump());
/*
[  1;
   4;
   7]
*/
m2.put(0, 0, 55);
System.out.println(m2.dump());
/*
[ 55;
   4;
   7]
*/
System.out.println(m.dump());
/*
[ 55,   2,   3;
   4,   5,   6;
   7,   8,   9]
*/
```

- `colRange()` — возвращает матрицу, содержащую элементы из диапазона столбцов. Изменение значений в этой матрице затронет и значения исходной матрицы. Форматы метода:

```
public Mat colRange(int startcol, int endcol)
public Mat colRange(Range r)
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
//Mat m2 = m.colRange(1, 2);
Mat m2 = m.colRange(new Range(1, 2));
System.out.println(m2.dump());
```

```
/*
[  2;
   5;
   8]
*/
m2.put(0, 0, 55);
System.out.println(m2.dump());
/*
[ 55;
   5;
   8]
*/
System.out.println(m.dump());
/*
[  1,  55,   3;
   4,   5,   6;
   7,   8,   9]
*/
```

- `diag()` — возвращает матрицу, содержащую элементы, расположенные по диагонали в исходной матрице. Изменение значений в этой матрице затронет и значения исходной матрицы. Форматы метода:

```
public Mat diag()
public Mat diag(int d)
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
Mat m2 = m.diag();
System.out.println(m2.dump());
/*
[  1;
   5;
   9]
*/
m2.put(0, 0, 55);
System.out.println(m2.dump());
/*
[ 55;
   5;
   9]
*/
```

```
System.out.println(m.dump());
/*
[ 55,   2,   3;
   4,   5,   6;
   7,   8,   9]
*/
```

Параметр `d` задает смещение относительно главной диагонали. В предыдущем примере этот параметр равен `0`. Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8SC1);
m.diag(0).setTo(new Scalar(0));
m.diag(1).setTo(new Scalar(1));
m.diag(2).setTo(new Scalar(2));
m.diag(-1).setTo(new Scalar(-1));
m.diag(-2).setTo(new Scalar(-2));
System.out.println(m.dump());
/*
[  0,   1,   2;
  -1,   0,   1;
  -2,  -1,   0]
*/
```

- `submat()` — возвращает матрицу, содержащую элементы из прямоугольного диапазона. Изменение значений в этой матрице затронет и значения исходной матрицы. Форматы метода:

```
public Mat submat(int rowStart, int rowEnd,
                  int colStart, int colEnd)
public Mat submat(Range rowRange, Range colRange)
public Mat submat(Rect roi)
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
Mat m2 = m.submat(new Rect(0, 0, 2, 2));
// Mat m2 = m.submat(0, 2, 0, 2);
// Mat m2 = m.submat(new Range(0, 2), new Range(0, 2));
System.out.println(m2.dump());
/*
[  1,   2;
   4,   5]
*/
```

```
m2.put(0, 0, 55);
System.out.println(m2.dump());
/*
[ 55,   2;
   4,   5]
*/
System.out.println(m.dump());
/*
[ 55,   2,   3;
   4,   5,   6;
   7,   8,   9]
*/
```

Для получения фрагмента матрицы можно воспользоваться следующими конструкторами класса `Mat`:

```
Mat(Mat m, Range rowRange)
Mat(Mat m, Range rowRange, Range colRange)
Mat(Mat m, Rect roi)
```

Изменение значений в этой матрице затронет и значения исходной матрицы. Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
System.out.println(m.dump());
/*
[  1,   2,   3;
   4,   5,   6;
   7,   8,   9]
*/
Mat m2 = new Mat(m, new Range(0, 2));
m2.setTo(new Scalar(5));
System.out.println(m2.dump());
/*
[  5,   5,   5;
   5,   5,   5]
*/
System.out.println(m.dump());
/*
[  5,   5,   5;
   5,   5,   5;
   7,   8,   9]
*/
```

Параметр `rowRange` задает диапазон для строк, а параметр `colRange` — для столбцов:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
System.out.println(m.dump());
/*
[  1,   2,   3;
   4,   5,   6;
   7,   8,   9]
*/
Mat m2 = new Mat(m, new Range(0, 2), new Range(0, 2));
// Mat m2 = new Mat(m, new Rect(0, 0, 2, 2));
m2.setTo(new Scalar(5));
System.out.println(m2.dump());
/*
[  5,   5;
   5,   5]
*/
System.out.println(m.dump());
/*
[  5,   5,   3;
   5,   5,   6;
   7,   8,   9]
*/
```

> ***Примечание***
>
> Создать субматрицу с измененной структурой исходной матрицы можно с помощью метода `reshape()` (см. *разд. 2.1.8*).

Все рассмотренные в этом разделе методы возвращают лишь ссылку на диапазон (субматрицу), который называют *областью интереса* (или *ROI*). Изменение значений в этой субматрице затронет и значения исходной матрицы. Проверить, с чем мы имеем дело: с матрицей или субматрицей — позволяет метод `isSubmatrix()`. Метод возвращает значение `true`, если мы имеем дело с субматрицей. Формат метода:

```
public boolean isSubmatrix()
```

Пример:

```
Mat m = new Mat(2, 3, CvType.CV_8UC1, new Scalar(1));
System.out.println(m.isSubmatrix());  // false
Mat m2 = m.row(0);
System.out.println(m2.isSubmatrix()); // true
```

2.1.6. Создание копии матрицы

Для создания полной копии матрицы предназначен метод `clone()`. Формат метода:

```
public Mat clone()
```

Пример:

```
Mat m = new Mat(2, 3, CvType.CV_8UC1, new Scalar(1));
System.out.println(m.dump());
/*
[  1,   1,   1;
   1,   1,   1]
*/
Mat m2 = m.clone();
m2.setTo(new Scalar(5));
System.out.println(m2.dump());
/*
[  5,   5,   5;
   5,   5,   5]
*/
System.out.println(m.dump());
/*
[  1,   1,   1;
   1,   1,   1]
*/
```

Скопировать одну матрицу в другую позволяет метод `copyTo()`. Форматы метода:

```
public void copyTo(Mat m)
public void copyTo(Mat m, Mat mask)
```

При использовании первого формата матрица `m` будет иметь точно такие же элементы (их копию), как и исходная матрица:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1, new Scalar(1));
Mat m2 = new Mat(2, 2, CvType.CV_8UC1, new Scalar(5));
m2.copyTo(m);
m.put(0, 0, 11);
System.out.println(m2.dump());
/*
[  5,   5;
   5,   5]
*/
System.out.println(m.dump());
/*
[ 11,   5;
   5,   5]
*/
```

Второй формат позволяет дополнительно наложить маску. Ненулевые значения маски указывают, какие элементы будут скопированы. Матрица маски должна иметь тип `CV_8U`. Пример:

```
Mat m = new Mat(2, 2, CvType.CV_8UC1, new Scalar(1));
Mat m2 = new Mat(2, 2, CvType.CV_8UC1, new Scalar(5));
Mat mask = Mat.eye(2, 2, CvType.CV_8UC1);
m2.copyTo(m, mask);
System.out.println(m2.dump());
/*
[  5,   5;
   5,   5]
*/
System.out.println(m.dump());
/*
[  5,   1;
   1,   5]
*/
```

Если матрица перераспределяется (например, размеры матриц не совпадают), то матрица перед копированием инициализируется нулями. Пример:

```
Mat m = new Mat(2, 2, CvType.CV_8UC1, new Scalar(1));
Mat m2 = new Mat(3, 3, CvType.CV_8UC1, new Scalar(5));
Mat mask = Mat.eye(3, 3, CvType.CV_8UC1);
m2.copyTo(m, mask);
System.out.println(m2.dump());
/*
[  5,   5,   5;
   5,   5,   5;
   5,   5,   5]
*/
System.out.println(m.dump());
/*
[  5,   0,   0;
   0,   5,   0;
   0,   0,   5]
*/
```

Скопировать элементы матрицы `m` в конец исходной матрицы позволяет метод `push_back()`. Обратите внимание, количество столбцов у матриц должно совпадать. Формат метода:

```
public void push_back(Mat m)
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
m.put(0, 0,
      1, 2, 3,
```

```
      4, 5, 6,
      7, 8, 9
);
Mat m2 = new Mat(1, 3, CvType.CV_8UC1);
m2.put(0, 0,
      10, 11, 12);
m.push_back(m2);
System.out.println(m.dump());
/*
[  1,   2,   3;
   4,   5,   6;
   7,   8,   9;
  10,  11,  12]
*/
```

2.1.7. Транспонирование матрицы

С помощью метода `t()` можно создать *транспонированную матрицу* (строки станут столбцами). Формат метода:

```
public Mat t()
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
Mat m2 = m.t();
System.out.println(m.dump());
/*
[  1,   2,   3;
   4,   5,   6;
   7,   8,   9]
*/
System.out.println(m2.dump());
/*
[  1,   4,   7;
   2,   5,   8;
   3,   6,   9]
*/
```

Для создания транспонированной матрицы можно также воспользоваться статическим методом `transpose()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void transpose(Mat src, Mat dst)
```

В первом параметре указывается исходная матрица, а во втором — матрица, в которую будет записан результат операции.

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
m.put(0, 0,
      1, 2, 3,
      4, 5, 6,
      7, 8, 9
);
Mat m2 = new Mat();
Core.transpose(m, m2);
System.out.println(m2.dump());
/*
[  1,   4,   7;
   2,   5,   8;
   3,   6,   9]
*/
```

2.1.8. Изменение структуры матрицы

Создать субматрицу с измененной структурой исходной матрицы позволяет метод `reshape()`. Форматы метода:

```
public Mat reshape(int cn)
public Mat reshape(int cn, int rows)
```

В параметре `cn` задается новое количество каналов. Если указать значение `0`, то количество каналов не изменится. В параметре `rows` можно указать новое количество строк.

Обратите внимание: метод возвращает ссылку на диапазон (субматрицу), а не на копию матрицы. Никакие данные не копируются. Помните также, что при изменении структуры общее количество элементов должно совпадать. Нельзя добавить какие-либо элементы или удалить существующие. Иными словами, если мы имеем матрицу, содержащую 18 элементов, то в параметре `rows` можно указать число, на которое 18 делится без остатка. Например, `1`, `2`, `3`, `6` и т. д., но не `4` и `5`.

Пример:

```
Mat m = new Mat(2, 3, CvType.CV_8UC3);
m.put(0, 0,
       1,  2,  3,  4,  5,  6,
       7,  8,  9, 10, 11, 12,
      13, 14, 15, 16, 17, 18
);
System.out.println(m.channels()); // 3
System.out.println(m.size());     // 3x2
System.out.println(m.dump());
```

```
/*
[  1,   2,   3,   4,   5,   6,   7,   8,   9;
  10,  11,  12,  13,  14,  15,  16,  17,  18]
*/
Mat m2 = m.reshape(1);
System.out.println(m2.channels()); // 1
System.out.println(m2.size());     // 9x2
System.out.println(m2.dump());
/*
[  1,   2,   3,   4,   5,   6,   7,   8,   9;
  10,  11,  12,  13,  14,  15,  16,  17,  18]
*/
Mat m3 = m.reshape(1, m.rows() * m.cols() * m.channels());
System.out.println(m3.channels()); // 1
System.out.println(m3.size());     // 1x18
// Создаем транспонированную копию
Mat m3_copy = m3.t();
System.out.println(m3_copy.dump());
// [  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
//   13,  14,  15,  16,  17,  18]
Mat m4 = m.reshape(1, m.rows() * m.cols());
System.out.println(m4.channels()); // 1
System.out.println(m4.size());     // 3x6
System.out.println(m4.dump());
/*
[  1,   2,   3;
   4,   5,   6;
   7,   8,   9;
  10,  11,  12;
  13,  14,  15;
  16,  17,  18]
*/
m4.put(0, 0, 33); // Изменения коснутся исходной матрицы !!!
Mat m5 = m.reshape(1, 1);
System.out.println(m5.channels()); // 1
System.out.println(m5.size());     // 18x1
System.out.println(m5.dump());
// Метод reshape() возвращает субматрицу, а не копию матрицы !!!
// [ 33,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
//   13,  14,  15,  16,  17,  18]
```

2.1.9. Удаление матрицы

Для удаления матрицы предназначен метод `release()`. Он не удаляет матрицу напрямую, а всего лишь уменьшает количество ссылок на единицу. Данные удаляются только тогда, когда количество ссылок станет равно нулю. Формат метода:

```
public void release()
```

Пример:

```
Mat m = new Mat(10, 10, CvType.CV_8UC1, new Scalar(1));
System.out.println(m);
// Mat [ 10*10*CV_8UC1, isCont=true, isSubmat=false,
// nativeObj=0x19cc8d70, dataAddr=0x19cc9bc0 ]
m.release();
System.out.println(m);
// Mat [ 0*0*CV_8UC1, isCont=true, isSubmat=false,
// nativeObj=0x19cc8d70, dataAddr=0x0 ]
m = null;
```

Обратите внимание на размеры и значение параметра `dataAddr` после вызова метода `release()`. Параметр стал указывать в никуда (нулевой указатель). Данные удалились, но нативный объект остался. Чтобы его удалить, достаточно присвоить переменной значение `null`. Если переменная выйдет из области видимости, то нативный объект будет удален автоматически.

2.2. Изменение значений матрицы

Для изменения различных характеристик цвета — например, яркости, нужно уметь правильно выполнять операции присваивания, прибавления и вычитания значений, их умножения и деления. Первое, что можно сразу придумать, — создать цикл, внутри него поэлементно получать значения с помощью метода `get()`, изменять значения, а потом присваивать их обратно с помощью метода `put()`. В большинстве случаев это неправильный способ. Работать он будет, но очень не эффективно! Правильный способ заключается в максимальном использовании специализированных методов из библиотеки OpenCV. Чтобы вы поняли, насколько велика бывает разница, выполним две операции разными способами. Вначале присвоим всем элементам одинаковое значение, а затем прибавим к этому значению новое значение. При этом измерим время выполнения (листинг 2.1).

Листинг 2.1. Сравнение способов изменения значений

```
Mat m = new Mat(1000, 1000, CvType.CV_8UC4);
long n = System.currentTimeMillis();
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, 10, 20, 30, 40);
   }
}
System.out.println("put() - " + (System.currentTimeMillis() - n));

n = System.currentTimeMillis();
m.setTo(new Scalar(10, 20, 30, 40));
System.out.println("setTo() - " + (System.currentTimeMillis() - n));
```

```
n = System.currentTimeMillis();
double[] arr = null;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      arr = m.get(i, j);
      arr[0] += 10;
      arr[1] += 10;
      arr[2] += 10;
      arr[3] += 10;
      m.put(i, j, arr);
   }
}
System.out.println("get() - " + (System.currentTimeMillis() - n));

n = System.currentTimeMillis();
Core.add(m, new Scalar(10, 10, 10, 10), m);
System.out.println("add() - " + (System.currentTimeMillis() - n));
```

Результат выполнения программы в версии 3.3.0 (время в миллисекундах):

```
put() - 125
setTo() - 1
get() - 250
add() - 543
```

Результат в версии 2.4.13.0 (и в версии 2.4.13.3-vc14):

```
put() - 125
setTo() - 1
get() - 232
add() - 1
```

Итак, присваивание значения внутри цикла выполняется за 125 миллисекунд, а с помощью метода `setTo()` всего за одну! Прибавление значений внутри цикла выполняется за 232 миллисекунды, а с помощью метода `add()` в версии 2.4.13 всего за одну! Время выполнения на вашем компьютере может отличаться от времени на моем компьютере. Да это и не важно. Важно процентное соотношение. Помните, что специализированные методы часто выполняют те же самые операции в 100 и порой даже более чем в 200 раз быстрее!

К сожалению, результат может отличаться в разных версиях OpenCV. Обратите внимание на скорость выполнения метода `add()` в версии 3. Метод выполняется даже дольше, чем при использовании в цикле методов `get()` и `put()`. Однако если мы увеличим количество элементов:

```
Mat m = new Mat(4000, 4000, CvType.CV_8UC4);
```

то метод `add()` будет уже быстрее методов `get()` и `put()`. Результат в версии 3.3.0:

```
put() - 1540
setTo() - 29
```

```
get() - 3610
add() - 616
```

Результат в версии 2.4.13 значительно лучше:

```
put() - 1632
setTo() - 41
get() - 3654
add() - 14
```

Как я заметил, многие методы в версии 3 на моем компьютере (Windows 8 *x*64) работают очень медленно. Причем **только при первом вызове** — как будто ждут чего-то! Например, если добавить еще один вызов метода `add()`, то первый вызов отработает медленно, а вот второй и все последующие не уступят по скорости версии 2.4.13. Пример:

```
Mat m = new Mat(4000, 4000, CvType.CV_8UC4, Scalar.all(20));
long n = System.currentTimeMillis();
Core.add(m, new Scalar(10, 10, 10, 10), m);
System.out.println("add() 1 - " + (System.currentTimeMillis() - n));
n = System.currentTimeMillis();
Core.add(m, new Scalar(10, 10, 10, 10), m);
System.out.println("add() 2 - " + (System.currentTimeMillis() - n));
```

Результат в версии 3.3.0:

```
add() 1 - 622
add() 2 - 14
```

Результат в версии 2.4.13:

```
add() 1 - 15
add() 2 - 15
```

В число таких методов входят: `eye()`, `transpose()`, `add()`, `subtract()`, `multiply()`, `divide()` и др. Похоже, эти методы как-то связаны друг с другом: первый вызов любого из этих методов делает последующие вызовы других методов быстрыми.

Пример:

```
Mat m = new Mat(4000, 4000, CvType.CV_8UC4, Scalar.all(20));
long n = System.currentTimeMillis();
Core.subtract(m, new Scalar(10, 10, 10, 10), m);
System.out.println("subtract() - " + (System.currentTimeMillis() - n));
n = System.currentTimeMillis();
Core.add(m, new Scalar(10, 10, 10, 10), m);
System.out.println("add() - " + (System.currentTimeMillis() - n));
```

Результат в версии 3.3.0:

```
subtract() - 561
add() - 14
```

Если методы поменять местами, то результат будет таким:

```
add() - 588
subtract() - 15
```

Вполне возможно, что на вашем компьютере такой проблемы с версией 3 и не будет. Попробуйте замерить время и только после этого делайте окончательные выводы.

Что же делать, если нужно выполнить операцию, для которой нет подходящего специализированного метода? Один из вариантов: скопировать данные из матрицы в массив, обработать массив, а затем скопировать данные из массива обратно в матрицу:

```
Mat m = new Mat(1000, 1000, CvType.CV_8UC4, new Scalar(10, 20, 30, 40));
long n = System.currentTimeMillis();
byte[] arr = new byte[m.rows() * m.cols() * m.channels()];
m.get(0, 0, arr);
int color = 0;
for (int i = 0, j = arr.length; i < j; i++) {
   color = arr[i] & 0xFF;
   color = color + 10;
   color = color > 255 ? 255 : color;
   arr[i] = (byte) color;
}
m.put(0, 0, arr);
System.out.println(System.currentTimeMillis() - n);
```

Обработка занимает от 5 до 16 миллисекунд. Да, это медленнее метода `add()` в версии 2.4.13, но в любой версии этот способ работает быстрее поэлементного перебора с помощью методов `get()` и `put()`.

При обработке элементов массива следует помнить о выходе значений за границы диапазона. Давайте рассмотрим пример:

```
int k = 200 + 100;
byte kb = (byte) k;
System.out.println( kb & 0xFF ); // 44
```

В этом примере число `300` выходит за пределы диапазона `0...255`. В итоге, после приведения к диапазону типа `byte` от `-128` до `127`, получим переполнение и абсолютно неправильный результат `44` вместо `255`. Проверку выхода за границы диапазона специализированные методы библиотеки OpenCV производят автоматически, а вот при работе с массивом нужно выполнить проверку и обработку самостоятельно:

```
int k = 200 + 100;
// int k = -10;
k = k > 255 ? 255 : (k < 0 ? 0 : k);
System.out.println( k );          // 255
byte kb = (byte) k;
System.out.println( kb & 0xFF ); // 255
```

2.2.1. Изменение сразу всех значений

Задать значение сразу всем элементам матрицы позволяет метод `setTo()` из класса `Mat`. Форматы метода:

```
public Mat setTo(Scalar value)
public Mat setTo(Scalar value, Mat mask)
public Mat setTo(Mat value)
public Mat setTo(Mat value, Mat mask)
```

Пример:

```
// import org.opencv.core.MatOfInt;
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
System.out.println(m.dump());
// [  1,   2,   3,   4,   1,   2,   3,   4]
m.setTo(new Scalar(0, 128, 200, 218));
System.out.println(m.dump());
// [  0, 128, 200, 218,   0, 128, 200, 218]
m.setTo(new MatOfInt(5, 6, 7, 8));
System.out.println(m.dump());
// [  5,   6,   7,   8,   5,   6,   7,   8]
```

В параметре `mask` можно указать маску. Матрица с маской должна иметь такие же размеры, как и исходная матрица, и содержать один канал. В исходную матрицу будут вставлены только те значения `value`, которые соответствуют значению больше `0` в маске. Если элемент в маске имеет значение `0`, то значение `value` в исходную матрицу не вставляется.

Пример:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Mat mask = new Mat(1, 2, CvType.CV_8UC1);
mask.put(0, 0, 0, 1);
System.out.println(m.dump());
// [  1,   2,   3,   4,   1,   2,   3,   4]
System.out.println(mask.dump()); // [  0,   1]
m.setTo(new Scalar(0, 128, 200, 218), mask);
System.out.println(m.dump());
// [  1,   2,   3,   4,   0, 128, 200, 218]
```

С помощью метода `convertTo()` из класса `Mat` можно не только преобразовать тип матрицы, но и изменить значения всех компонентов цвета. Форматы метода:

```
public void convertTo(Mat m, int rtype)
public void convertTo(Mat m, int rtype, double alpha)
public void convertTo(Mat m, int rtype, double alpha, double beta)
```

В первом параметре указываем ссылку на матрицу, в которую будут скопированы данные, а во втором параметре — новый тип. Если во втором параметре указать значение `-1`, то тип будет соответствовать типу исходной матрицы. Значение пара-

метра alpha будет умножено на значение компонента цвета, а значение beta — прибавлено. Умножим все значения на 2:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
m.convertTo(m, -1, 2, 0);
System.out.println(m.toString());
// Mat [ 1*2*CV_8UC4, isCont=true, isSubmat=false,
// nativeObj=0x3712b0, dataAddr=0x373d80 ]
System.out.println(m.dump());
// [  2,   4,   6,   8,   2,   4,   6,   8]
```

Прибавим значение 10:

```
m.convertTo(m, -1, 1, 10);
System.out.println(m.dump());
// [ 12,  14,  16,  18,  12,  14,  16,  18]
```

Разделим на 2 (обратите внимание: одно из значений при делении обязательно должно иметь тип double, иначе будет выполнено целочисленное деление, при котором остаток отбрасывается):

```
m.convertTo(m, -1, 1.0 / 2, 0);
System.out.println(m.dump());
// [  6,   7,   8,   9,   6,   7,   8,   9]
```

Теперь вычтем 10 и прибавим 300:

```
m.convertTo(m, -1, 1, -10);
System.out.println(m.dump());
// [  0,   0,   0,   0,   0,   0,   0,   0]
m.convertTo(m, -1, 1, 300);
System.out.println(m.dump());
// [255, 255, 255, 255, 255, 255, 255, 255]
```

Обратите внимание на итоговые значения в последнем примере. Если результат выходит за границы диапазона допустимых значений для типа, то значение будет равно минимальному (в нашем случае значение 0) или максимальному (в нашем случае 255) допустимому значению.

Для изменения значений каналов можно также воспользоваться статическим методом transform() из класса Core. Формат метода:

```
import org.opencv.core.Core;
public static void transform(Mat src, Mat dst, Mat m)
```

В первом параметре указывается исходная матрица, во втором — матрица, в которую будет записан результат, а в третьем — матрица трансформации (с вещественными значениями, например, матрица типа CV_32F).

В качестве примера поменяем местами первый и третий каналы:

```
Mat m = new Mat(1, 3, CvType.CV_8UC3);
m.put(0, 0,
      1, 2, 3,
```

```
      4, 5, 6,
      7, 8, 9
);
Mat kernel = new Mat(3, 3, CvType.CV_32FC1);
double b1 = 0, g1 = 0, r1 = 1;
double b2 = 0, g2 = 1, r2 = 0;
double b3 = 1, g3 = 0, r3 = 0;
kernel.put(0, 0,
      b1, g1, r1, // blue  = b * b1 + g * g1 + r * r1
      b2, g2, r2, // green = b * b2 + g * g2 + r * r2
      b3, g3, r3  // red   = b * b3 + g * g3 + r * r3
);
Mat m2 = new Mat();
Core.transform(m, m2, kernel);
System.out.println(m2.dump());
// [  3,   2,   1,   6,   5,   4,   9,   8,   7]
```

2.2.2. Сложение

Прибавить какое-либо значение ко всем элементам матрицы позволяет статический метод `add()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void add(Mat src1, Scalar src2, Mat dst)
public static void add(Mat src1, Scalar src2, Mat dst, Mat mask)
public static void add(Mat src1, Scalar src2, Mat dst, Mat mask,
                       int dtype)
public static void add(Mat src1, Mat src2, Mat dst)
public static void add(Mat src1, Mat src2, Mat dst, Mat mask)
public static void add(Mat src1, Mat src2, Mat dst, Mat mask,
                       int dtype)
```

В первом параметре указывается исходная матрица, во втором параметре — скаляр или другая матрица (значения которых прибавляются), а в третьем — матрица, в которую будут записаны данные. В параметре `mask` можно указать маску. Матрица с маской должна иметь такие же размеры, как и исходная матрица, и содержать один канал. В исходную матрицу будут вставлены только те значения, которые соответствуют значению больше `0` в маске. Если элемент в маске имеет значение `0`, то значение в исходную матрицу не вставляется. Параметр `dtype` позволяет указать тип матрицы с результатом (значение по умолчанию: `-1`).

Пример добавления значений:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Core.add(m, new Scalar(10, 20, 30, 40), m);
System.out.println(m.dump());
// [ 11,  22,  33,  44,  11,  22,  33,  44]
Core.add(m, new Scalar(300, 0, 0, 0), m);
System.out.println(m.dump());
```

```
// [255,  22,  33,  44, 255,  22,  33,  44]
Core.add(m, new Scalar(-300, -300, -300, -300), m);
System.out.println(m.dump());
// [  0,   0,   0,   0,   0,   0,   0,   0]

Mat m2 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Mat m3 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(4, 3, 2, 1));
Core.add(m2, m3, m);
System.out.println(m.dump());
// [  5,   5,   5,   5,   5,   5,   5,   5]
```

Статический метод `addWeighted()` из класса `Core` выполняет следующую операцию сложения:

```
dst(i) = src1(i) * alpha + src2(i) * beta + gamma;
```

Форматы метода:

```
import org.opencv.core.Core;
public static void addWeighted(Mat src1, double alpha, Mat src2,
                        double beta, double gamma, Mat dst)
public static void addWeighted(Mat src1, double alpha, Mat src2,
                        double beta, double gamma, Mat dst, int dtype)
```

Пример:

```
Mat m = new Mat(1, 4, CvType.CV_8UC1);
m.put(0, 0, 1, 2, 3, 4);
Mat m2 = new Mat(1, 4, CvType.CV_8UC1);
m2.put(0, 0, 5, 6, 7, 8);
System.out.println(m.dump());  // [  1,   2,   3,   4]
System.out.println(m2.dump()); // [  5,   6,   7,   8]
Mat m3 = new Mat();
Core.addWeighted(m, 2, m2, 1, 10.1, m3, CvType.CV_32F);
System.out.println(m3.dump()); // [17.1, 20.1, 23.1, 26.1]
Mat m4 = new Mat();
Core.addWeighted(m, 2, m2, 1, 300, m4);
System.out.println(m4.dump()); // [255, 255, 255, 255]
```

Статический метод `scaleAdd()` из класса `Core` выполняет следующую операцию сложения:

```
dst(i) = src1(i) * alpha + src2(i);
```

Формат метода:

```
import org.opencv.core.Core;
public static void scaleAdd(Mat src1, double alpha, Mat src2, Mat dst)
```

Пример:

```
Mat m = new Mat(1, 4, CvType.CV_8UC1);
m.put(0, 0, 1, 2, 3, 4);
Mat m2 = new Mat(1, 4, CvType.CV_8UC1);
```

```
m2.put(0, 0, 5, 6, 7, 8);
System.out.println(m.dump());  // [  1,   2,   3,   4]
System.out.println(m2.dump()); // [  5,   6,   7,   8]
Mat m3 = new Mat();
Core.scaleAdd(m, 2, m2, m3);
System.out.println(m3.dump()); // [  7,  10,  13,  16]
Mat m4 = new Mat();
Core.scaleAdd(m, 100, m2, m4);
System.out.println(m4.dump()); // [105, 206, 255, 255]
```

2.2.3. Вычитание

Вычесть какое-либо значение из значений элементов матрицы позволяет статический метод `subtract()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void subtract(Mat src1, Scalar src2, Mat dst)
public static void subtract(Mat src1, Scalar src2, Mat dst, Mat mask)
public static void subtract(Mat src1, Scalar src2, Mat dst, Mat mask,
                            int dtype)
public static void subtract(Mat src1, Mat src2, Mat dst)
public static void subtract(Mat src1, Mat src2, Mat dst, Mat mask)
public static void subtract(Mat src1, Mat src2, Mat dst, Mat mask,
                            int dtype)
```

В первом параметре указывается исходная матрица, во втором параметре — скаляр или другая матрица (значения из которой вычитаются), а в третьем — матрица, в которую будут записаны данные. В параметре `mask` можно указать маску. Матрица с маской должна иметь такие же размеры, как и исходная матрица, и содержать один канал. В исходную матрицу будут вставлены только те значения, которые соответствуют значению больше `0` в маске. Если элемент в маске имеет значение `0`, то значение в исходную матрицу не вставляется. Параметр `dtype` позволяет указать тип матрицы с результатом (значение по умолчанию: `-1`).

Пример вычитания значений:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(10, 20, 30, 40));
Core.subtract(m, new Scalar(1, 2, 3, 4), m);
System.out.println(m.dump());
// [  9,  18,  27,  36,   9,  18,  27,  36]
Core.subtract(m, new Scalar(300, 300, 300, 300), m);
System.out.println(m.dump());
// [  0,   0,   0,   0,   0,   0,   0,   0]

Mat m2 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Mat m3 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 9));
Core.subtract(m2, m3, m);
System.out.println(m.dump());
// [  0,   0,   0,   0,   0,   0,   0,   0]
```

С помощью статического метода `absdiff()` из класса `Core` можно вычислить абсолютную разницу. Вначале вычисляется разница, затем знак отбрасывается и в конце проверяется выход за диапазон значений. При использовании метода `subtract()` отрицательное значение будет заменено нулем, тогда как при использовании метода `absdiff()` оно станет положительным и будет ограничено максимальным значением диапазона, а не минимальным. Форматы метода:

```
import org.opencv.core.Core;
public static void absdiff(Mat src1, Scalar src2, Mat dst)
public static void absdiff(Mat src1, Mat src2, Mat dst)
```

Пример:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(10, 20, 30, 40));
Core.absdiff(m, new Scalar(1, 2, 3, 4), m);
System.out.println(m.dump());
// [  9,  18,  27,  36,   9,  18,  27,  36]
Core.absdiff(m, new Scalar(300, 300, 300, 300), m);
System.out.println(m.dump());
// [255, 255, 255, 255, 255, 255, 255, 255]

Mat m2 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Mat m3 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 9));
Core.absdiff(m2, m3, m);
System.out.println(m.dump());
// [  0,   0,   0,   5,   0,   0,   0,   5]
```

2.2.4. Умножение

Умножить все элементы матрицы на какое-либо значение позволяет статический метод `multiply()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void multiply(Mat src1, Scalar src2, Mat dst)
public static void multiply(Mat src1, Scalar src2, Mat dst, double scale)
public static void multiply(Mat src1, Scalar src2, Mat dst, double scale,
                            int dtype)
public static void multiply(Mat src1, Mat src2, Mat dst)
public static void multiply(Mat src1, Mat src2, Mat dst, double scale)
public static void multiply(Mat src1, Mat src2, Mat dst, double scale,
                            int dtype)
```

В первом параметре указывается исходная матрица, во втором параметре — скаляр или другая матрица, а в третьем — матрица, в которую будут записаны данные. В параметре `scale` можно указать дополнительный коэффициент масштаба. Параметр `dtype` позволяет указать тип матрицы с результатом (значение по умолчанию: `-1`).

Пример:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Core.multiply(m, new Scalar(2, 2, 2, 2), m);
System.out.println(m.dump());
// [  2,   4,   6,   8,   2,   4,   6,   8]
Core.multiply(m, new Scalar(2, 2, 2, 2), m, 2);
System.out.println(m.dump());
// [  8,  16,  24,  32,   8,  16,  24,  32]
Core.multiply(m, new Scalar(2, 2, 2, 2), m, 20);
System.out.println(m.dump());
// [255, 255, 255, 255, 255, 255, 255, 255]

Mat m2 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Mat m3 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(4, 3, 2, 1));
Core.multiply(m2, m3, m);
System.out.println(m.dump());
// [  4,   6,   6,   4,   4,   6,   6,   4]
```

Для перемножения двух матриц можно также воспользоваться методом `mul()` из класса `Mat`. Форматы метода:

```
public Mat mul(Mat m)
public Mat mul(Mat m, double scale)
```

Значения исходной матрицы будут умножены на значения матрицы, указанной в первом параметре, и на коэффициент масштаба `scale` (значение по умолчанию: `1`). Метод возвращает матрицу с результатом. Исходная матрица не изменяется.

Пример:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Mat m2 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 2, 3, 4));
Mat m3 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(4, 3, 2, 1));
m = m2.mul(m3);
System.out.println(m.dump());
// [  4,   6,   6,   4,   4,   6,   6,   4]
m = m2.mul(m3, 2);
System.out.println(m.dump());
// [  8,  12,  12,   8,   8,  12,  12,   8]
```

2.2.5. Деление

Разделить все элементы матрицы на какое-либо значение позволяет статический метод `divide()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void divide(Mat src1, Scalar src2, Mat dst)
public static void divide(Mat src1, Scalar src2, Mat dst, double scale)
public static void divide(Mat src1, Scalar src2, Mat dst, double scale,
                          int dtype)
```

```
public static void divide(Mat src1, Mat src2, Mat dst)
public static void divide(Mat src1, Mat src2, Mat dst, double scale)
public static void divide(Mat src1, Mat src2, Mat dst, double scale,
                          int dtype)
public static void divide(double scale, Mat src2, Mat dst)
public static void divide(double scale, Mat src2, Mat dst, int dtype)
```

В параметре `src1` указывается исходная матрица, в параметре `src2` — скаляр или другая матрица (на значения которых производится деление), а в параметре `dst` — матрица, в которую будут записаны данные. В параметре `scale` можно указать дополнительный коэффициент масштаба, который умножается на значение `src1`. Параметр `dtype` позволяет указать тип матрицы с результатом (значение по умолчанию: `-1`).

Пример:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(10, 20, 30, 40));
Mat m1 = new Mat(1, 2, CvType.CV_8UC4);
Core.divide(m, new Scalar(2, 2, 2, 2), m1);
System.out.println(m1.dump());
// [  5,  10,  15,  20,   5,  10,  15,  20]
Core.divide(m, new Scalar(2, 2, 2, 2), m1, 2);
System.out.println(m1.dump());
// [ 10,  20,  30,  40,  10,  20,  30,  40]
Core.divide(m, new Scalar(100, 100, 100, 100), m1);
System.out.println(m1.dump());
// [  0,   0,   0,   0,   0,   0,   0,   0]

Mat m2 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(10, 20, 30, 40));
Mat m3 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(5, 2, 3, 4));
Core.divide(m2, m3, m);
System.out.println(m.dump());
// [  2,  10,  10,  10,   2,  10,  10,  10]
```

2.2.6. Вычисление квадратного корня

Вычислить квадратный корень позволяет статический метод `sqrt()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void sqrt(Mat src, Mat dst)
```

В первом параметре указывается исходная матрица, а во втором — матрица, в которую будут записаны данные. Матрицы должны иметь глубину `CV_32F` или `CV_64F`. Проверка выхода за границы диапазона не производится.

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_64FC4, new Scalar(100, 16, 9, 4));
Mat m2 = new Mat(1, 1, CvType.CV_64FC4);
Core.sqrt(m, m2);
System.out.println(m2.dump()); // [10, 4, 3, 2]
```

2.2.7. Возведение в степень

Возвести значения матрицы в степень позволяет статический метод `pow()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void pow(Mat src, double power, Mat dst)
```

В первом параметре указывается исходная матрица, во втором — степень, а в третьем — матрица, в которую будут записаны данные.

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_64FC4, new Scalar(2, 3, 4, 5));
Mat m2 = new Mat(1, 1, CvType.CV_64FC4);
Core.pow(m, 2, m2);
System.out.println(m2.dump()); // [4, 9, 16, 25]
```

2.2.8. Вычисление натурального логарифма и экспоненты

Вычислить натуральный логарифм позволяет статический метод `log()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void log(Mat src, Mat dst)
```

В первом параметре указывается исходная матрица, во втором — матрица, в которую будут записаны данные. Матрицы должны иметь глубину `CV_32F` или `CV_64F`. Проверка выхода за границы диапазона не производится.

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_32FC4, new Scalar(1, 2, 2.7, 7.29));
Mat m2 = new Mat(1, 1, CvType.CV_32FC4);
Core.log(m, m2);
System.out.println(m2.dump());
// [0, 0.69314718, 0.9932518, 1.9865035]
```

Вычислить экспоненту позволяет статический метод `exp()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void exp(Mat src, Mat dst)
```

2.2.9. Использование таблицы соответствия

Если нужно изменить значения матрицы не на одинаковую величину, а на произвольные значения, например, вычисленные по формулам, то для матриц типа `CV_8U` и `CV_8S` можно воспользоваться таблицей соответствия. Таблица соответствия состоит из 256 элементов и может содержать несколько каналов. Индекс позиции в этой таблице соответствует старому значению элемента матрицы, а значение —

новому значению. Например, мы вычислили, что значение 0 нужно заменить значением 255, следовательно, в элемент таблицы соответствия с индексом 0 нужно положить значение 255. После применения таблицы соответствия к матрице все элементы матрицы, имевшие значение 0, изменят свое значение на 255.

Применить таблицу соответствия к матрице позволяет статический метод LUT() из класса Core. Формат метода:

```
import org.opencv.core.Core;
public static void LUT(Mat src, Mat lut, Mat dst)
```

В первом параметре указывается исходная матрица, во втором — матрица с таблицей соответствия, а в третьем — матрица, в которую будут записаны новые значения.

В качестве примера инвертируем значения матрицы:

```
Mat m = new Mat(1, 5, CvType.CV_8UC1);
m.put(0, 0, 0, 64, 128, 192, 255);
System.out.println(m.dump()); // [  0,  64, 128, 192, 255]

// Создание таблицы соответствия
Mat lut = new Mat(1, 256, CvType.CV_8UC1);
byte[] arr = new byte[256];
for (int i = 0; i < 256; i++) {
   arr[i] = (byte) (255 - i);
}
lut.put(0, 0, arr);
System.out.println(lut.dump());

// Преобразование в соответствии с таблицей
Core.LUT(m, lut, m);
System.out.println(m.dump()); // [255, 191, 127,  63,   0]
```

Если нужно изменять значения только отдельных каналов, то таблица соответствия должна состоять из нескольких каналов. Например, инвертируем значения только первого и третьего каналов, а значение второго оставим без изменений:

```
Mat m = new Mat(1, 2, CvType.CV_8UC3);
m.put(0, 0, 0, 128, 255, 64, 127, 192);
System.out.println(m.dump()); // [  0, 128, 255,  64, 127, 192]

// Создание таблицы соответствия
Mat lut = new Mat(1, 256, CvType.CV_8UC3);
byte[] arr = new byte[256 * 3];
for (int i = 0, j = 256 * 3, k = 0; i < j; i += 3, k++) {
   arr[i] = (byte) (255 - k);
   arr[i + 1] = (byte) k;
   arr[i + 2] = (byte) (255 - k);
}
```

```
lut.put(0, 0, arr);
System.out.println(lut.dump());

// Преобразование в соответствии с таблицей
Core.LUT(m, lut, m);
System.out.println(m.dump()); // [255, 128,   0, 191, 127,  63]
```

2.2.10. Нормализация и вычисление нормы

Произвести нормализацию матрицы позволяет статический метод `normalize()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void normalize(Mat src, Mat dst)
public static void normalize(Mat src, Mat dst, double alpha,
                             double beta, int norm_type)
public static void normalize(Mat src, Mat dst, double alpha,
                             double beta, int norm_type, int dtype)
public static void normalize(Mat src, Mat dst, double alpha,
                             double beta, int norm_type, int dtype,
                             Mat mask)
```

В первом параметре указывается исходная матрица, а во втором — матрица, в которую будет записан результат операции. Глубину итоговой матрицы можно указать в параметре `dtype`. Если указать значение `-1` (значение по умолчанию), то глубина итоговой матрицы будет соответствовать глубине исходной матрицы. В параметре `norm_type` задается тип нормализации (значение по умолчанию: `NORM_L2`). Доступны следующие типы (константы из класса `Core`):

```
public static final int NORM_INF
public static final int NORM_L1
public static final int NORM_L2
public static final int NORM_MINMAX
```

Выведем значения констант:

```
System.out.println(Core.NORM_INF);    // 1
System.out.println(Core.NORM_L1);     // 2
System.out.println(Core.NORM_L2);     // 4
System.out.println(Core.NORM_MINMAX); // 32
```

Параметр `alpha` задает норму (при использовании типов `NORM_INF`, `NORM_L1` и `NORM_L2`) или минимальное значение диапазона (при использовании типа `NORM_MINMAX`), а параметр `beta` — максимальное значение диапазона (только для типа `NORM_MINMAX`).

Пример нормализации диапазона:

```
Mat m = new Mat(1, 6, CvType.CV_64FC1);
m.put(0, 0, -50, -3, 0, 20, 250, 300);
System.out.println(m.dump()); // [-50, -3, 0, 20, 250, 300]
```

```
Mat m2 = new Mat();
Core.normalize(m, m2, 0, 255, Core.NORM_MINMAX, CvType.CV_64F);
System.out.println(m2.dump());
// [0, 34.24285714285714, 36.42857142857142,
// 50.99999999999999, 218.5714285714286, 255]
```

Вычислить абсолютную норму, а также абсолютную или относительную норму разницы двух матриц позволяет статический метод `norm()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static double norm(Mat src1)
public static double norm(Mat src1, int normType)
public static double norm(Mat src1, int normType, Mat mask)
public static double norm(Mat src1, Mat src2)
public static double norm(Mat src1, Mat src2, int normType)
public static double norm(Mat src1, Mat src2, int normType, Mat mask)
```

В параметрах `src1` и `src2` указываются исходные матрицы. Параметр `normType` задает способ вычисления нормы (значение по умолчанию: `NORM_L2`). Доступны следующие способы (константы из класса `Core`):

```
public static final int NORM_INF
public static final int NORM_L1
public static final int NORM_L2
public static final int NORM_RELATIVE
```

Если указаны только первые три способа, то вычисляется абсолютная норма, если нужно вычислить относительную норму, то следует дополнительно указать флаг `NORM_RELATIVE`.

2.2.11. Побитовые операции

Выполнить операции с отдельными битами позволяют следующие статические методы из класса `Core`:

- `bitwise_not()` — двоичная инверсия. Форматы метода:

  ```
  import org.opencv.core.Core;
  public static void bitwise_not(Mat src, Mat dst)
  public static void bitwise_not(Mat src, Mat dst, Mat mask)
  ```

  Пример:

  ```
  Mat m = new Mat(1, 1, CvType.CV_32SC1);
  m.put(0, 0, 100);
  System.out.printf("%32s\n", Integer.toBinaryString(100));
  //                          1100100
  Mat result = new Mat();
  Core.bitwise_not(m, result);
  System.out.printf("%32s\n", Integer.toBinaryString(
                    (int) result.get(0, 0)[0]));
  // 11111111111111111111111110011011
  ```

- `bitwise_and()` — двоичное И. Форматы метода:

```
public static void bitwise_and(Mat src1, Mat src2, Mat dst)
public static void bitwise_and(Mat src1, Mat src2, Mat dst,
                               Mat mask)
```

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_32SC1, new Scalar(100));
Mat m2 = new Mat(1, 1, CvType.CV_32SC1, new Scalar(75));
System.out.printf("%32s\n", Integer.toBinaryString(100));
//                           1100100
System.out.printf("%32s\n", Integer.toBinaryString(75));
//                           1001011
Mat result = new Mat();
Core.bitwise_and(m, m2, result);
System.out.printf("%32s\n", Integer.toBinaryString(
                  (int) result.get(0, 0)[0]));
//                           1000000
```

- `bitwise_or()` — двоичное ИЛИ. Форматы метода:

```
public static void bitwise_or(Mat src1, Mat src2, Mat dst)
public static void bitwise_or(Mat src1, Mat src2, Mat dst,
                              Mat mask)
```

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_32SC1, new Scalar(100));
Mat m2 = new Mat(1, 1, CvType.CV_32SC1, new Scalar(75));
System.out.printf("%32s\n", Integer.toBinaryString(100));
//                           1100100
System.out.printf("%32s\n", Integer.toBinaryString(75));
//                           1001011
Mat result = new Mat();
Core.bitwise_or(m, m2, result);
System.out.printf("%32s\n", Integer.toBinaryString(
                  (int) result.get(0, 0)[0]));
//                           1101111
```

- `bitwise_xor()` — двоичное исключающее ИЛИ. Форматы метода:

```
public static void bitwise_xor(Mat src1, Mat src2, Mat dst)
public static void bitwise_xor(Mat src1, Mat src2, Mat dst,
                               Mat mask)
```

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_32SC1, new Scalar(100));
Mat m2 = new Mat(1, 1, CvType.CV_32SC1, new Scalar(250));
System.out.printf("%32s\n", Integer.toBinaryString(100));
//                           1100100
```

```
System.out.printf("%32s\n", Integer.toBinaryString(250));
//                          11111010
Mat result = new Mat();
Core.bitwise_xor(m, m2, result);
System.out.printf("%32s\n", Integer.toBinaryString(
                  (int) result.get(0, 0)[0]));
//                          10011110
```

2.3. Поиск минимального, максимального и среднего значений

Выполнить поиск минимальных значений позволяет статический метод `min()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void min(Mat src1, Scalar src2, Mat dst)
public static void min(Mat src1, Mat src2, Mat dst)
```

Для поиска максимальных значений предназначен статический метод `max()` из класса `Core`. Форматы метода:

```
public static void max(Mat src1, Scalar src2, Mat dst)
public static void max(Mat src1, Mat src2, Mat dst)
```

В первом параметре указывается исходная матрица, во втором — скаляр или матрица, а в третьем — матрица, в которую будут записаны данные.

Пример:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4, new Scalar(2, 3, 4, 5));
Mat m2 = new Mat(1, 2, CvType.CV_8UC4, new Scalar(1, 5, 3, 7));
Mat m3 = new Mat(1, 2, CvType.CV_8UC4);

Core.min(m, new Scalar(1, 5, 3, 7), m3);
System.out.println(m3.dump());
// [  1,   3,   3,   5,   1,   3,   3,   5]
Core.max(m, new Scalar(1, 5, 3, 7), m3);
System.out.println(m3.dump());
// [  2,   5,   4,   7,   2,   5,   4,   7]

Core.min(m, m2, m3);
System.out.println(m3.dump());
// [  1,   3,   3,   5,   1,   3,   3,   5]
Core.max(m, m2, m3);
System.out.println(m3.dump());
// [  2,   5,   4,   7,   2,   5,   4,   7]
```

Найти максимальное и минимальное значения в матрице, состоящей из одного канала, позволяет статический метод `minMaxLoc()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core.MinMaxLocResult;
public static Core.MinMaxLocResult minMaxLoc(Mat src)
public static Core.MinMaxLocResult minMaxLoc(Mat src, Mat mask)
```

Метод возвращает объект с максимальным (свойство `maxVal`) и минимальным (свойство `minVal`) значениями. С помощью свойств `maxLoc` и `minLoc` можно получить позицию этих значений в матрице.

Пример:

```
// import java.util.ArrayList;
Mat m = new Mat(1, 2, CvType.CV_8UC4);
m.put(0, 0, 2, 3, 4, 8);
m.put(0, 1, 8, 3, 9, 5);
ArrayList<Mat> list = new ArrayList<Mat>();
Core.split(m, list); // Разделение на отдельные каналы

for (Mat mm : list) {
   MinMaxLocResult r = Core.minMaxLoc(mm);
   System.out.println(r.maxVal + " " + r.minVal + " " +
                      r.maxLoc + " " + r.minLoc);
}
/*
8.0 2.0 {1.0, 0.0} {0.0, 0.0}
3.0 3.0 {0.0, 0.0} {0.0, 0.0}
9.0 4.0 {1.0, 0.0} {0.0, 0.0}
8.0 5.0 {0.0, 0.0} {1.0, 0.0}
*/
```

Вычислить среднее значение для всех каналов матрицы позволяет статический метод `mean()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static Scalar mean(Mat src)
public static Scalar mean(Mat src, Mat mask)
```

Пример:

```
Mat m = new Mat(1, 2, CvType.CV_8UC4);
m.put(0, 0, 2, 3, 4, 5);
m.put(0, 1, 8, 3, 9, 5);
System.out.println(Core.mean(m)); // [5.0, 3.0, 6.5, 5.0]
```

Чтобы получить минимальное, максимальное или среднее значения для каждого столбца или строки матрицы следует воспользоваться статическим методом `reduce()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void reduce(Mat src, Mat dst, int dim, int rtype)
public static void reduce(Mat src, Mat dst, int dim, int rtype,
                          int dtype)
```

В первом параметре указывается исходная матрица, а во втором — матрица, в которую будет записан результат. Если в параметре `dim` указать значение `0`, то значения станут считаться по столбцам, и результирующая матрица будет содержать одну строку. Если в параметре `dim` указать значение `1`, то значения станут считаться по строкам, и результирующая матрица будет содержать один столбец. В параметре `rtype` можно указать следующие константы из класса `Core`:

- `REDUCE_MIN` — минимальное значение. Формат:

```
public static final int REDUCE_MIN
```

- `REDUCE_MAX` — максимальное значение. Формат:

```
public static final int REDUCE_MAX
```

- `REDUCE_AVG` — среднее значение. Формат:

```
public static final int REDUCE_AVG
```

- `REDUCE_SUM` — сумма значений. Формат:

```
public static final int REDUCE_SUM
```

Выведем значения констант:

```
System.out.println(Core.REDUCE_MIN); // 3
System.out.println(Core.REDUCE_MAX); // 2
System.out.println(Core.REDUCE_AVG); // 1
System.out.println(Core.REDUCE_SUM); // 0
```

В параметре `dtype` можно указать тип матрицы с результатом. Если указать значение `-1`, то матрица с результатом будет иметь тот же тип, что и исходная матрица.

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
m.put(0, 0,
      1, 2, 9,
      6, 5, 4,
      7, 8, 3
);
Mat result = new Mat();
// Минимальные значения по столбцам
Core.reduce(m, result, 0, Core.REDUCE_MIN);
System.out.println(result.dump()); // [  1,   2,   3]
// Минимальные значения по строкам
Core.reduce(m, result, 1, Core.REDUCE_MIN);
System.out.println(result.dump());
/*
[  1;
   4;
   3]*/
// Максимальные значения по столбцам
Core.reduce(m, result, 0, Core.REDUCE_MAX);
System.out.println(result.dump()); // [  7,   8,   9]
```

```
// Максимальные значения по строкам
Core.reduce(m, result, 1, Core.REDUCE_MAX);
System.out.println(result.dump());
/*
[  9;
   6;
   8]*/
// Средние значения по столбцам
Core.reduce(m, result, 0, Core.REDUCE_AVG, CvType.CV_32F);
System.out.println(result.dump()); // [4.666667, 5, 5.3333335]
// Средние значения по строкам
Core.reduce(m, result, 1, Core.REDUCE_AVG, CvType.CV_32F);
System.out.println(result.dump());
/*
[4;
 5;
 6]*/
```

2.4. Вычисление суммы элементов

Посчитать сумму элементов матрицы позволяет статический метод `sumElems()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static Scalar sumElems(Mat src)
```

Метод возвращает объект класса `Scalar` с суммами значений по каждому каналу.

Пример:

```
Mat m = new Mat(1, 3, CvType.CV_8UC3);
m.put(0, 0,
      1, 2, 3,
      4, 5, 6,
      7, 8, 9
);
Scalar sum = Core.sumElems(m);
System.out.println(sum); // [12.0, 15.0, 18.0, 0.0]
```

Для подсчета суммы значений по диагонали можно воспользоваться статическим методом `trace()` из класса `Core`. Формат метода:

```
public static Scalar trace(Mat src)
```

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
m.put(0, 0,
      1, 2, 10,
      4, 5, 6,
      7, 8, 9
);
```

```
Scalar sum = Core.trace(m);
System.out.println(sum); // [15.0, 0.0, 0.0, 0.0]
```

Чтобы получить сумму значений для каждого столбца или строки матрицы, следует воспользоваться статическим методом `reduce()` из класса `Core` (описание метода см. в *разд. 2.3*).

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
m.put(0, 0,
      1, 2, 9,
      6, 5, 4,
      7, 8, 3
);
Mat sum = new Mat();
// Сумма значений по столбцам
Core.reduce(m, sum, 0, Core.REDUCE_SUM, CvType.CV_32F);
System.out.println(sum.dump()); // [14, 15, 16]
// Сумма значений по строкам
Core.reduce(m, sum, 1, Core.REDUCE_SUM, CvType.CV_32F);
System.out.println(sum.dump());
/*
[12;
 15;
 18]*/
```

2.5. Заполнение матрицы случайными числами

Заполнить матрицу случайными числами позволяют статические методы `randu()` и `randn()` из класса `Core`. Форматы методов:

```
import org.opencv.core.Core;
public static void randu(Mat dst, double low, double high)
public static void randn(Mat dst, double mean, double stddev)
```

В первом параметре указывается ссылка на матрицу, в которую будут вставлены случайные значения. Параметр `low` в методе `randu()` задает минимальное значение (включительно), а параметр `high` — максимальное значение (исключая данное значение). Параметр `mean` в методе `randn()` задает среднее значение, а параметр `stddev` — стандартное отклонение.

Настроить генератор случайных чисел на новую последовательность позволяет статический метод `setRNGSeed()` из класса `Core`. Формат метода:

```
public static void setRNGSeed(int seed) // Нет в версии 2.4
```

Пример:

```
Core.setRNGSeed(200);
Mat m = new Mat(1, 2, CvType.CV_8UC4);
```

```
Core.randu(m, 0, 256);
System.out.println(m.dump());
// [208,  95, 133, 241, 225,  46, 115,  23]
Core.randu(m, 0, 256);
System.out.println(m.dump());
// [222, 217,  52, 225, 129, 192, 176,   0]

Mat m2 = new Mat(1, 2, CvType.CV_32FC1);
Core.randn(m2, 0, 1);
System.out.println(m2.dump()); // [0.0014689101, -0.29025102]
```

Перемешать элементы матрицы случайным образом позволяет статический метод `randShuffle()` из класса `Core`. Элементы внутри каналов при этом не изменяют своих позиций. Форматы метода:

```
public static void randShuffle(Mat dst)
public static void randShuffle(Mat dst, double iterFactor)
```

Пример:

```
Core.setRNGSeed(200);
Mat m = new Mat(1, 8, CvType.CV_8UC1);
m.put(0, 0, 1, 2, 3, 4, 5, 6, 7, 8);
Core.randShuffle(m);
System.out.println(m.dump());
// [  1,   3,   7,   6,   2,   8,   4,   5]
Core.randShuffle(m, 1.);
System.out.println(m.dump());
// [  4,   5,   2,   8,   1,   6,   7,   3]
```

2.6. Сортировка элементов матрицы

Отсортировать элементы матрицы позволяет статический метод `sort()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void sort(Mat src, Mat dst, int flags)
```

В первом параметре указывается исходная матрица, содержащая один канал, а во втором — матрица, в которую будет записан результат. В параметре `flags` можно указать комбинацию следующих флагов (константы из класса `Core`):

- `SORT_ASCENDING` — сортировка по возрастанию. Формат:

  ```
  public static final int SORT_ASCENDING
  ```

- `SORT_DESCENDING` — сортировка по убыванию. Формат:

  ```
  public static final int SORT_DESCENDING
  ```

- `SORT_EVERY_ROW` — сортировка элементов по строкам. Формат:

  ```
  public static final int SORT_EVERY_ROW
  ```

- SORT_EVERY_COLUMN — сортировка элементов по столбцам. Формат:

  public static final int SORT_EVERY_COLUMN

Флаги SORT_ASCENDING и SORT_DESCENDING, а также SORT_EVERY_ROW и SORT_EVERY_COLUMN являются взаимоисключающими. Выведем значения констант:

```
System.out.println(Core.SORT_ASCENDING);    // 0
System.out.println(Core.SORT_DESCENDING);   // 16
System.out.println(Core.SORT_EVERY_ROW);    // 0
System.out.println(Core.SORT_EVERY_COLUMN); // 1
```

Пример сортировки матрицы:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
m.put(0, 0,
      1, 2, 9,
      6, 5, 4,
      7, 8, 3
);
Mat m2 = new Mat();
// Сортировка по строкам
Core.sort(m, m2, Core.SORT_ASCENDING | Core.SORT_EVERY_ROW);
System.out.println(m2.dump());
/*
[  1,   2,   9;
   4,   5,   6;
   3,   7,   8]*/
Core.sort(m, m2, Core.SORT_DESCENDING | Core.SORT_EVERY_ROW);
System.out.println(m2.dump());
/*
[  9,   2,   1;
   6,   5,   4;
   8,   7,   3]*/
// Сортировка по столбцам
Core.sort(m, m2, Core.SORT_ASCENDING | Core.SORT_EVERY_COLUMN);
System.out.println(m2.dump());
/*
[  1,   2,   3;
   6,   5,   4;
   7,   8,   9]*/
Core.sort(m, m2, Core.SORT_DESCENDING | Core.SORT_EVERY_COLUMN);
System.out.println(m2.dump());
/*
[  7,   8,   9;
   6,   5,   4;
   1,   2,   3]*/
```

Помимо метода sort() существует статический метод sortIdx(). Формат метода:

```
public static void sortIdx(Mat src, Mat dst, int flags)
```

Смысл всех параметров точно такой же, как и у метода `sort()`. Отличие между методами в том, что метод `sort()` возвращает матрицу с отсортированными элементами, а метод `sortIdx()` — матрицу с индексами элементов.

Пример:

```
Mat m = new Mat(3, 3, CvType.CV_8UC1);
m.put(0, 0,
      1, 2, 9,
      6, 5, 4,
      7, 8, 3
);
Mat m2 = new Mat();
// Сортировка по строкам
Core.sortIdx(m, m2, Core.SORT_ASCENDING | Core.SORT_EVERY_ROW);
System.out.println(m2.dump());
/*
[0, 1, 2;
 2, 1, 0;
 2, 0, 1]*/
// Сортировка по столбцам
Core.sortIdx(m, m2, Core.SORT_ASCENDING | Core.SORT_EVERY_COLUMN);
System.out.println(m2.dump());
/*
[0, 0, 2;
 1, 1, 1;
 2, 2, 0]*/
```

2.7. Сравнение элементов

Сравнить элементы позволяет статический метод `compare()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void compare(Mat src1, Scalar src2, Mat dst, int cmpop)
public static void compare(Mat src1, Mat src2, Mat dst, int cmpop)
```

Параметр `cmpop` задает способ сравнения. Можно указать следующие константы из класса `Core`:

```
public static final int CMP_EQ // src1 == src2
public static final int CMP_NE // src1 != src2
public static final int CMP_GT // src1 >  src2
public static final int CMP_GE // src1 >= src2
public static final int CMP_LT // src1 <  src2
public static final int CMP_LE // src1 <= src2
```

Выведем значения констант:

```
System.out.println(Core.CMP_EQ); // 0
System.out.println(Core.CMP_NE); // 5
```

```
System.out.println(Core.CMP_GT); // 1
System.out.println(Core.CMP_GE); // 2
System.out.println(Core.CMP_LT); // 3
System.out.println(Core.CMP_LE); // 4
```

Если сравнение вернуло значение `true`, то соответствующий элемент в итоговой матрице `dst` получит значение `255`.

Пример:

```
Mat m = new Mat(1, 4, CvType.CV_8UC1);
m.put(0, 0, 1, 2, 3, 4);
Mat m2 = new Mat(1, 4, CvType.CV_8UC1);
m2.put(0, 0, 1, 8, 3, 4);
Mat result = new Mat(1, 4, CvType.CV_8UC1);
Core.compare(m, m2, result, Core.CMP_EQ); // ==
System.out.println(result.dump()); // [255,   0, 255, 255]
Core.compare(m, m2, result, Core.CMP_NE); // !=
System.out.println(result.dump()); // [  0, 255,   0,   0]
Core.compare(m, new Scalar(3), result, Core.CMP_GT); // >
System.out.println(result.dump()); // [  0,   0,   0, 255]
Core.compare(m, new Scalar(3), result, Core.CMP_GE); // >=
System.out.println(result.dump()); // [  0,   0, 255, 255]
Core.compare(m, new Scalar(3), result, Core.CMP_LT); // <
System.out.println(result.dump()); // [255, 255,   0,   0]
Core.compare(m, new Scalar(3), result, Core.CMP_LE); // <=
System.out.println(result.dump()); // [255, 255, 255,   0]
```

2.8. Прочие методы

Статический метод `countNonZero()` из класса `Core` позволяет посчитать количество элементов, значение которых не равно нулю. Формат метода:

```
import org.opencv.core.Core;
public static int countNonZero(Mat src)
```

Матрица, указываемая в качестве параметра, должна содержать один канал.

Пример:

```
Mat m = new Mat(1, 4, CvType.CV_8UC1);
m.put(0, 0, 1, 2, 0, 4);
System.out.println(m.dump());             // [  1,   2,   0,   4]
System.out.println(Core.countNonZero(m)); // 3
```

Получить индексы элементов, значение которых не равно нулю, позволяет статический метод `findNonZero()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void findNonZero(Mat src, Mat idx)
```

Матрица, указанная в первом параметре, должна иметь тип `CV_8UC1`.

Пример:

```
Mat m = new Mat(1, 4, CvType.CV_8UC1);
m.put(0, 0, 1, 2, 0, 4);
System.out.println(m.dump());   // [  1,   2,   0,   4]
Mat idx = new Mat();
Core.findNonZero(m, idx);
System.out.println(idx.dump());
/*
[0, 0;
 1, 0;
 3, 0]*/
```

Заменить все значения `NaN` указанным значением `val` (значение по умолчанию: `0`) позволяет статический метод `patchNaNs()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void patchNaNs(Mat a)
public static void patchNaNs(Mat a, double val)
```

Пример:

```
Mat m = new Mat(1, 4, CvType.CV_32FC1);
m.put(0, 0, 1, 2, 0, 0.0 / 0);
System.out.println(m.dump());   // [1, 2, 0, nan]
Core.patchNaNs(m, 0);
System.out.println(m.dump());   // [1, 2, 0, 0]
```

Проверить значения элементов матрицы можно с помощью статического метода `checkRange()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static boolean checkRange(Mat a)
public static boolean checkRange(Mat a, boolean quiet,
                                 double minVal, double maxVal)
```

Метод вернет значение `false`, если существует хотя бы один элемент, равный бесконечности, или `NaN` или значение элемента не входит в диапазон между `minVal` и `maxVal`. Если в параметре `quiet` указать значение `false`, то вместо возврата значения `false` будет генерироваться исключение. По умолчанию параметр `quiet` имеет значение `true`.

Пример:

```
Mat m = new Mat(1, 4, CvType.CV_32FC1);
m.put(0, 0, 1, 2, 0, 0.0 / 0);
System.out.println(m.dump());           // [1, 2, 0, nan]
System.out.println(Core.checkRange(m)); // false
Mat m2 = new Mat(1, 4, CvType.CV_32FC1);
m2.put(0, 0, 1, 2, 0, 20);
System.out.println(Core.checkRange(m2, true, 0, 50)); // true
```

2.9. Классы *MatOfByte*, *MatOfInt*, *MatOfFloat* и *MatOfDouble*

Класс `Mat` наследуют классы `MatOfByte`, `MatOfInt`, `MatOfFloat` и `MatOfDouble`. Эти классы реализуют одномерные матрицы соответствующего типа данных, элементы которых содержат только один канал. Инструкции импорта:

```
import org.opencv.core.MatOfByte;       // CV_8U
import org.opencv.core.MatOfInt;        // CV_32S
import org.opencv.core.MatOfFloat;      // CV_32F
import org.opencv.core.MatOfDouble;     // CV_64F
```

Кроме того, существуют классы `MatOfInt4`, `MatOfFloat4` и `MatOfFloat6`, которые могут содержать несколько каналов. Инструкции импорта:

```
import org.opencv.core.MatOfInt4;       // CV_32SC4
import org.opencv.core.MatOfFloat4;     // CV_32FC4
import org.opencv.core.MatOfFloat6;     // CV_32FC6
```

Конструкторы класса `MatOfInt`:

```
MatOfInt()
MatOfInt(int... a)
MatOfInt(Mat m)
```

Конструкторы других классов точно такие же, только отличается тип данных.

Пример:

```
int[] arr = {1, 2, 3};
MatOfInt m1 = new MatOfInt();
MatOfInt m2 = new MatOfInt(arr);

System.out.println(m1.dump()); // []
System.out.println(m2.dump());
/*
[1;
 2;
 3]*/
```

Методы класса `MatOfInt`:

- `fromArray()` — заполняет матрицу значениями из массива. Формат метода:

  ```
  public void fromArray(int... a)
  ```

  Пример:

  ```
  int[] arr = {1, 2, 3};
  MatOfInt m = new MatOfInt();
  m.fromArray(arr);
  System.out.println(m.dump());
  ```

```
/*
[1;
 2;
 3]*/
```

- `fromList()` — заполняет матрицу значениями из списка. Формат метода:

```
public void fromList(List<Integer> lb)
```

Пример:

```
// import java.util.ArrayList;
// import java.util.Collections;
ArrayList<Integer> list = new ArrayList<Integer>();
Collections.addAll(list, 1, 2, 3);
MatOfInt m = new MatOfInt();
m.fromList(list);
System.out.println(m.dump());
/*
[1;
 2;
 3]*/
```

- `toArray()` — возвращает массив со значениями из матрицы. Формат метода:

```
public int[] toArray()
```

Пример:

```
// import java.util.Arrays;
MatOfInt m = new MatOfInt(1, 2, 3);
int[] arr = m.toArray();
System.out.println(Arrays.toString(arr)); // [1, 2, 3]
```

- `toList()` — возвращает список со значениями из матрицы. Формат метода:

```
public List<Integer> toList()
```

Пример:

```
// import java.util.List;
MatOfInt m = new MatOfInt(1, 2, 3);
List<Integer> list = m.toList();
System.out.println(list); // [1, 2, 3]
```

Методы других классов точно такие же, только отличается тип данных.

Для создания матрицы из списка можно воспользоваться следующими статическими методами из класса `Converters`:

```
import org.opencv.utils.Converters;
public static Mat vector_char_to_Mat(List<Byte> bs)      // CV_8SC1
public static Mat vector_uchar_to_Mat(List<Byte> bs)     // CV_8UC1
public static Mat vector_int_to_Mat(List<Integer> is)    // CV_32SC1
public static Mat vector_float_to_Mat(List<Float> fs)    // CV_32FC1
public static Mat vector_double_to_Mat(List<Double> ds)  // CV_64FC1
```

Пример:

```
ArrayList<Integer> list = new ArrayList<Integer>();
Collections.addAll(list, 1, 2, 3);
Mat m = Converters.vector_int_to_Mat(list);
System.out.println(m.dump());
/*
[1;
 2;
 3]*/
```

Выполнить обратную операцию позволяют следующие статические методы из класса `Converters`:

```
// m.type() == CV_8SC1; m.cols() == 1
public static void Mat_to_vector_char(Mat m, List<Byte> bs)
// m.type() == CV_8UC1; m.cols() == 1
public static void Mat_to_vector_uchar(Mat m, List<Byte> us)
// m.type() == CV_32SC1; m.cols() == 1
public static void Mat_to_vector_int(Mat m, List<Integer> is)
// m.type() == CV_32FC1; m.cols() == 1
public static void Mat_to_vector_float(Mat m, List<Float> fs)
// m.type() == CV_64FC1; m.cols() == 1
public static void Mat_to_vector_double(Mat m, List<Double> ds)
```

Пример:

```
ArrayList<Integer> list = new ArrayList<Integer>();
MatOfInt m = new MatOfInt(1, 2, 3);
Converters.Mat_to_vector_int(m, list);
System.out.println(list); // [1, 2, 3]
```

2.10. Классы *MatOfPoint*, *MatOfPoint3* и *MatOfRect*

Класс `Mat` наследуют также классы `MatOfPoint`, `MatOfPoint2f`, `MatOfPoint3`, `MatOfPoint3f`, `MatOfRect` и `MatOfRect2d`. Инструкции импорта:

```
import org.opencv.core.MatOfPoint;       // CV_32SC2
import org.opencv.core.MatOfPoint2f;     // CV_32FC2
import org.opencv.core.MatOfPoint3;      // CV_32SC3
import org.opencv.core.MatOfPoint3f;     // CV_32FC3
import org.opencv.core.MatOfRect;        // CV_32SC4
// Класса MatOfRect2d нет в версии 2.4
import org.opencv.core.MatOfRect2d;      // CV_64FC4
```

Конструкторы и методы этих классов ничем не отличаются от рассмотренных в предыдущем разделе — различия только в типах данных.

Пример:

```
MatOfPoint m = new MatOfPoint();
m.fromArray(new Point(0, 0), new Point(1, 1));
System.out.println(m.dump());
/*
[0, 0;
 1, 1]*/
Point[] arr = m.toArray();
System.out.println(Arrays.toString(arr)); // [{0.0, 0.0}, {1.0, 1.0}]

ArrayList<Point> list = new ArrayList<Point>();
Collections.addAll(list, new Point(0, 0), new Point(1, 1));
MatOfPoint m2 = new MatOfPoint();
m2.fromList(list);
System.out.println(m2.dump());
/*
[0, 0;
 1, 1]*/
List<Point> list2 = m2.toList();
System.out.println(list2); // [{0.0, 0.0}, {1.0, 1.0}]
```

Обратите внимание: класс `MatOfPoint2f` принимает и возвращает объект класса `Point`, а не `Point2f`. Эту особенность можно использовать для преобразования матриц.

Пример:

```
MatOfPoint m = new MatOfPoint();
m.fromArray(new Point(0, 0), new Point(1, 1));
// MatOfPoint => MatOfPoint2f
MatOfPoint2f m2 = new MatOfPoint2f(m.toArray());
Core.add(m2, new Scalar(0.1, 0.6), m2);
System.out.println(m2.dump());
/*
[0.1, 0.60000002;
 1.1, 1.6]*/
// MatOfPoint2f => MatOfPoint
MatOfPoint m3 = new MatOfPoint(m2.toArray());
System.out.println(m3.dump());
/*
[0, 0;
 1, 1]*/
```

Для создания матрицы из списка можно воспользоваться следующими статическими методами из класса `Converters`:

```
import org.opencv.utils.Converters;
public static Mat vector_Point_to_Mat(List<Point> pts)   // CV_32S
public static Mat vector_Point2f_to_Mat(List<Point> pts) // CV_32F
```

```
public static Mat vector_Point2d_to_Mat(List<Point> pts) // CV_64F
public static Mat vector_Point_to_Mat(List<Point> pts, int typeDepth)
// typeDepth: CV_32S, CV_32F или CV_64F
public static Mat vector_Point3i_to_Mat(List<Point3> pts) // CV_32S
public static Mat vector_Point3f_to_Mat(List<Point3> pts) // CV_32F
public static Mat vector_Point3d_to_Mat(List<Point3> pts) // CV_64F
public static Mat vector_Point3_to_Mat(List<Point3> pts, int typeDepth)
// typeDepth: CV_32S, CV_32F или CV_64F
public static Mat vector_Rect_to_Mat(List<Rect> rs)     // CV_32SC4
// Метода vector_Rect2d_to_Mat() нет в версии 2.4
public static Mat vector_Rect2d_to_Mat(List<Rect2d> rs) // CV_64FC4
```

Пример:

```
ArrayList<Point> list = new ArrayList<Point>();
Collections.addAll(list, new Point(0, 0), new Point(1, 1));
Mat m = Converters.vector_Point_to_Mat(list);
System.out.println(m.dump());
/*
[0, 0;
 1, 1]*/
```

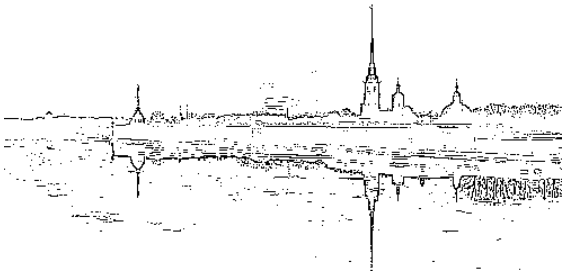
Выполнить обратную операцию позволяют следующие статические методы из класса `Converters`:

```
// m.type() == CV_32SC2, CV_32FC2 или CV_64FC2; m.cols() == 1
public static void Mat_to_vector_Point2f(Mat m, List<Point> pts)
public static void Mat_to_vector_Point2d(Mat m, List<Point> pts)
public static void Mat_to_vector_Point(Mat m, List<Point> pts)
// m.type() == CV_32SC3, CV_32FC3 или CV_64FC3; m.cols() == 1
public static void Mat_to_vector_Point3i(Mat m, List<Point3> pts)
public static void Mat_to_vector_Point3f(Mat m, List<Point3> pts)
public static void Mat_to_vector_Point3d(Mat m, List<Point3> pts)
public static void Mat_to_vector_Point3(Mat m, List<Point3> pts)
// m.type() == CV_32SC4; m.cols() == 1
public static void Mat_to_vector_Rect(Mat m, List<Rect> rs)
// Метода Mat_to_vector_Rect2d() нет в версии 2.4
// m.type() == CV_64FC4; m.cols() == 1
public static void Mat_to_vector_Rect2d(Mat m, List<Rect2d> rs)
```

Пример:

```
ArrayList<Point> list = new ArrayList<Point>();
MatOfPoint m = new MatOfPoint(new Point(0, 0), new Point(1, 1));
Converters.Mat_to_vector_Point(m, list);
System.out.println(list); // [{0.0, 0.0}, {1.0, 1.0}]
```

ГЛАВА 3



Создание и преобразование изображений

Изображение представляет собой двумерную таблицу, состоящую из столбцов (ширина изображения) и строк (высота изображения). Каждая ячейка такой таблицы, имеющая квадратную или прямоугольную форму, содержит описание цвета пиксела. В свою очередь, цвет может описываться несколькими компонентами. Черно-белое изображение и изображение в оттенках серого содержит всего один компонент, полноцветное — три компонента (обычно это красный, зеленый и синий — цветовая модель RGB), а полноцветное с альфа-каналом (с прозрачностью) — четыре компонента (обычно это красный, зеленый, синий и альфа-канал — цветовая модель RGBA).

Обратите внимание: в библиотеке OpenCV порядок следования компонентов цвета отличается от обычного представления. Вначале идет синий, затем зеленый, потом красный и последним указывается альфа-канал (цветовая модель BGRA). Таким образом, изображение есть не что иное, как матрица, каждый элемент которой содержит один, три или четыре канала. Для хранения изображений используется класс `Mat`, а для указания значений компонентов цвета — класс `Scalar`.

3.1. Загрузка изображения из файла

Для загрузки изображения из файла предназначен статический метод `imread()` из класса `Imgcodecs`. Форматы метода:

```
import org.opencv.imgcodecs.Imgcodecs;
public static Mat imread(String filename)
public static Mat imread(String filename, int flags)
```

В версии 2.4 метод `imread()` расположен в классе `Highgui`. Инструкция импорта:

```
import org.opencv.highgui.Highgui; // 2.4
```

В первом параметре указывается путь к файлу. Обратите внимание: в пути не должно быть русских букв, только латиница! В противном случае вы получите пустую матрицу, даже если путь существует. Перечислим поддерживаемые методом `imread()` форматы файлов:

- JPEG files — файлы с расширениями: jpeg, jpg, jpe;
- JPEG 2000 files — файлы с расширением jp2;
- Portable Network Graphics — файлы с расширением png;
- Windows bitmaps — файлы с расширениями: bmp, dib;
- TIFF files — файлы с расширениями: tiff, tif;
- Portable image format — файлы с расширениями: pbm, pgm, ppm;
- Sun rasters — файлы с расширениями: sr, ras.

В версии 3.3 доступны также форматы файлов с расширениями: pxm, pnm, webp, exr, hdr, pic и др.

Если изображение загрузить не удалось, то метод `empty()` из класса `Mat` вернет значение `true`. Загрузим изображение в формате JPEG и выведем информацию о нем:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto1.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
System.out.println(img.width());
System.out.println(img.height());
System.out.println(CvType.typeToString(img.type()));
System.out.println(img.channels());
```

Примерный вывод:

```
3482
2321
CV_8UC3
3
```

Первое значение представляет ширину изображения, второе — высоту, третье — тип матрицы, а четвертое — количество каналов.

Параметр `flags` позволяет дополнительно указать различные флаги. Если параметр имеет значение `-1`, то изображение загружается как есть (включая альфа-канал), если равен `0` — то будет выполнено преобразование в оттенки серого, если больше `0` — изображение будет иметь три канала (без альфа-канала). В качестве значения можно указать следующие статические константы (или их комбинацию) из класса `Imgcodecs`:

- `IMREAD_UNCHANGED` — изображение загружается как есть (включая альфа-канал). Формат:

  ```
  public static final int IMREAD_UNCHANGED
  ```

  В версии 2.4 используется следующая константа из класса `Highgui`:

  ```
  public static final int CV_LOAD_IMAGE_UNCHANGED
  ```

- `IMREAD_GRAYSCALE` — изображение преобразуется в оттенки серого. Формат:

  ```
  public static final int IMREAD_GRAYSCALE
  ```

В версии 2.4 используется следующая константа из класса `Highgui`:

```
public static final int CV_LOAD_IMAGE_GRAYSCALE
```

- `IMREAD_COLOR` — цветное изображение из трех каналов (если другие флаги не установлены, то по 8 битов на канал — тип `CV_8UC3`). Формат:

  ```
  public static final int IMREAD_COLOR
  ```

  В версии 2.4 используется следующая константа из класса `Highgui`:

  ```
  public static final int CV_LOAD_IMAGE_COLOR
  ```

- `IMREAD_ANYDEPTH` — если флаг установлен, то вернет 16-битное или 32-битное значение (при условии, что изображение имеет такую глубину цвета, в противном случае — 8-битное). Формат:

  ```
  public static final int IMREAD_ANYDEPTH
  ```

  В версии 2.4 используется следующая константа из класса `Highgui`:

  ```
  public static final int CV_LOAD_IMAGE_ANYDEPTH
  ```

- `IMREAD_ANYCOLOR` — если флаг установлен, то вернет изображение в любом доступном цветовом формате. Формат:

  ```
  public static final int IMREAD_ANYCOLOR
  ```

  В версии 2.4 используется следующая константа из класса `Highgui`:

  ```
  public static final int CV_LOAD_IMAGE_ANYCOLOR
  ```

В версии 3.3 доступны также следующие константы:

```
public static final int IMREAD_LOAD_GDAL
public static final int IMREAD_REDUCED_GRAYSCALE_2
public static final int IMREAD_REDUCED_GRAYSCALE_4
public static final int IMREAD_REDUCED_GRAYSCALE_8
public static final int IMREAD_REDUCED_COLOR_2
public static final int IMREAD_REDUCED_COLOR_4
public static final int IMREAD_REDUCED_COLOR_8
public static final int IMREAD_IGNORE_ORIENTATION
```

Если указаны константы `IMREAD_REDUCED_GRAYSCALE_*`, то изображение загружается, преобразуется в оттенки серого и уменьшается в 2, 4 или 8 раз. Если указаны константы `IMREAD_REDUCED_COLOR_*`, то загружается цветное изображение (без альфа-канала) и уменьшается в 2, 4 или 8 раз. Если указана константа `IMREAD_IGNORE_ORIENTATION`, то при наличии флага ориентации в EXIF изображение загружается без поворота.

Выведем значения констант в версии 3.3:

```
System.out.println(Imgcodecs.IMREAD_UNCHANGED);          // -1
System.out.println(Imgcodecs.IMREAD_GRAYSCALE);          // 0
System.out.println(Imgcodecs.IMREAD_COLOR);              // 1
System.out.println(Imgcodecs.IMREAD_ANYDEPTH);           // 2
```

```
System.out.println(Imgcodecs.IMREAD_ANYCOLOR);            // 4
System.out.println(Imgcodecs.IMREAD_LOAD_GDAL);           // 8
System.out.println(Imgcodecs.IMREAD_REDUCED_GRAYSCALE_2); // 16
System.out.println(Imgcodecs.IMREAD_REDUCED_GRAYSCALE_4); // 32
System.out.println(Imgcodecs.IMREAD_REDUCED_GRAYSCALE_8); // 64
System.out.println(Imgcodecs.IMREAD_REDUCED_COLOR_2);     // 17
System.out.println(Imgcodecs.IMREAD_REDUCED_COLOR_4);     // 33
System.out.println(Imgcodecs.IMREAD_REDUCED_COLOR_8);     // 65
System.out.println(Imgcodecs.IMREAD_IGNORE_ORIENTATION);  // 128
```

Выведем значения констант в версии 2.4:

```
System.out.println(Highgui.CV_LOAD_IMAGE_UNCHANGED);      // -1
System.out.println(Highgui.CV_LOAD_IMAGE_GRAYSCALE);      // 0
System.out.println(Highgui.CV_LOAD_IMAGE_COLOR);          // 1
System.out.println(Highgui.CV_LOAD_IMAGE_ANYDEPTH);       // 2
System.out.println(Highgui.CV_LOAD_IMAGE_ANYCOLOR);       // 4
```

В первом формате метода `imread()` параметр `flags` имеет значение `IMREAD_COLOR`. Обратите внимание, в этом случае глубина цвета будет 8 битов без альфа-канала. Если нужен альфа-канал, то в параметре `flags` нужно указать значение `IMREAD_UNCHANGED`.

Пример загрузки PNG-файла с альфа-каналом:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto4.png",
                           Imgcodecs.IMREAD_UNCHANGED);
```

Пример загрузки TIFF-файла (16 битов или 32 бита, без альфа-канала):

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto8.tif",
          Imgcodecs.IMREAD_ANYCOLOR | Imgcodecs.IMREAD_ANYDEPTH);
```

Пример загрузки и преобразования в оттенки серого:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto1.jpg",
                           Imgcodecs.IMREAD_GRAYSCALE);
```

3.2. Сохранение изображения в файл

Сохранить изображение в файл позволяет статический метод `imwrite()` из класса `Imgcodecs`. Форматы метода:

```
import org.opencv.imgcodecs.Imgcodecs;
public static boolean imwrite(String filename, Mat img)
public static boolean imwrite(String filename, Mat img, MatOfInt params)
```

В версии 2.4 метод `imwrite()` расположен в классе `Highgui`. Инструкция импорта:

```
import org.opencv.highgui.Highgui;
```

В первом параметре указывается путь к файлу (обратите внимание, в пути не должно быть русских букв, только латиница), а во втором — изображение. Формат

файла определяется по расширению. Поддерживаемые форматы точно такие же, как и у метода `imread()` (см. *разд. 3.1*). Если файл с таким именем уже существует, то он будет перезаписан. Метод `imwrite()` возвращает значение `true`, если изображение успешно сохранено.

> ***Обратите внимание!***
>
> Порядок следования каналов при сохранении изображения должен быть `BGRA`.

Загрузим изображение в формате JPEG, а затем сохраним его в формате PNG:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto1.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
boolean st = Imgcodecs.imwrite("C:\\book\\opencv\\_foto1.png", img);
if (!st) {
   System.out.println("Не удалось сохранить изображение");
}
```

Изображения, имеющие тип `CV_8U`, можно сохранить в любом доступном формате. Чтобы сохранить изображения, имеющие тип `CV_16U`, нужно использовать форматы PNG, JPEG 2000 или TIFF. Если нужно сохранить изображение с альфа-каналом, то следует использовать формат PNG.

В параметре `params` можно дополнительно указать следующие основные настройки (полный список настроек смотрите в документации):

- для формата JPEG — флаг `IMWRITE_JPEG_QUALITY` (в версии 2.4 `CV_IMWRITE_JPEG_QUALITY` из класса `Highgui`) и качество изображения в виде числа от `0` (минимальное, маленький размер файла) до `100` (максимальное, большой размер файла), значение по умолчанию — `95`.

  Пример сохранения изображения в максимальном качестве:

  ```
  Imgcodecs.imwrite("C:\\book\\opencv\\_foto_100.jpg", img,
          new MatOfInt(Imgcodecs.IMWRITE_JPEG_QUALITY , 100));
  ```

- для формата PNG — флаг `IMWRITE_PNG_COMPRESSION` (в версии 2.4 `CV_IMWRITE_PNG_COMPRESSION` из класса `Highgui`) и степень сжатия изображения в виде числа от `0` (без сжатия, большой размер файла) до `9` (максимальное, маленький размер файла), значение по умолчанию — `1`.

  Пример сохранения изображения в максимальном качестве:

  ```
  Imgcodecs.imwrite("C:\\book\\opencv\\_foto_0.png", img,
          new MatOfInt(Imgcodecs.IMWRITE_PNG_COMPRESSION, 0));
  ```

3.3. Преобразование матрицы в массив и обратно

Если в первых двух параметрах метода `get()` из класса `Mat` указать нулевые значения, а в третьем — передать ссылку на массив соответствующего размера, то значения из матрицы будут скопированы в массив. Обратите внимание: тип массива должен соответствовать типу элементов матрицы, иначе возникнет исключение. Метод вернет количество скопированных элементов.

Пример:

```
Mat m = new Mat(3, 2, CvType.CV_8UC1);
double n = 1.0;
for (int i = 0, r = m.rows(); i < r; i++) {
   for (int j = 0, c = m.cols(); j < c; j++) {
      m.put(i, j, n++);
   }
}
byte[] arr = new byte[m.channels() * m.cols() * m.rows()];
System.out.println(m.get(0, 0, arr));     // 6
System.out.println(Arrays.toString(arr)); // [1, 2, 3, 4, 5, 6]
```

Если в первых двух параметрах метода `put()` из класса `Mat` указать нулевые значения, а в третьем — массив, то значения из массива будут скопированы в матрицу:

```
Mat m = new Mat(3, 2, CvType.CV_8UC1);
byte[] barr = {1, 2, 3, 4, 5, 6};
m.put(0, 0, barr);
System.out.println(m.dump());
/*
[  1,   2;
   3,   4;
   5,   6]*/
```

Статический метод `imencode()` из класса `Imgcodecs` позволяет преобразовать изображение в какой-либо формат, но, в отличие от метода `imwrite()`, не сохраняет изображение в файл, а записывает его в буфер. Форматы метода:

```
import org.opencv.imgcodecs.Imgcodecs;
public static boolean imencode(String ext, Mat img, MatOfByte buf)
public static boolean imencode(String ext, Mat img, MatOfByte buf,
                               MatOfInt params)
```

В версии 2.4 метод `imencode()` расположен в классе `Highgui`. Инструкция импорта:

```
import org.opencv.highgui.Highgui;
```

В первом параметре указывается расширение, во втором — изображение, а в третьем — ссылка на буфер. Формат файла определяется по расширению. Поддерживаемые форматы точно такие же, как и у метода `imwrite()` (см. *разд. 3.2*). В пара-

метре `params` можно дополнительно указать качество для формата JPEG и степень сжатия для формата PNG, как и в методе `imwrite()` (см. *разд. 3.2*). Метод `imencode()` возвращает значение `true`, если изображение успешно преобразовано.

Загрузим изображение из файла, а затем преобразуем в формат JPEG и сохраним его в буфере:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto1.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
MatOfByte buf = new MatOfByte();
boolean st = Imgcodecs.imencode(".jpg", img, buf,
             new MatOfInt(Imgcodecs.IMWRITE_JPEG_QUALITY , 100));
if (!st) {
   System.out.println("Не удалось преобразовать изображение");
}
```

В дальнейшем, например, мы можем передать данные из буфера конструктору класса `Image` из JavaFX:

```
// import java.io.ByteArrayInputStream;
// import javafx.scene.image.Image;
Image im = new Image(new ByteArrayInputStream(buf.toArray()));
```

Обратите внимание: если вы задумали таким вот образом преобразовывать объект класса `Mat` в объект класса `Image`, то помните, что это будет работать, но очень уж медленно. В последующих разделах мы рассмотрим более быстрые способы преобразования изображений в объекты классов `java.awt.image.BufferedImage` и `javafx.scene.image.Image`.

С помощью статического метода `imdecode()` из класса `Imgcodecs` можно преобразовать данные из буфера в объект класса `Mat`. Формат метода:

```
import org.opencv.imgcodecs.Imgcodecs;
public static Mat imdecode(Mat buf, int flags)
```

В версии 2.4 метод `imdecode()` расположен в классе `Highgui`. Инструкция импорта:

```
import org.opencv.highgui.Highgui;
```

В первом параметре указываются данные из буфера, а во втором — дополнительные настройки. Параметр `flags` аналогичен одноименному параметру метода `imread()` (см. *разд. 3.1*).

Загрузим данные из буфера и обработаем ошибки:

```
Mat img2 = Imgcodecs.imdecode(buf, Imgcodecs.IMREAD_COLOR);
if (img2.empty()) {
   System.out.println("Не удалось загрузить изображение");
}
```

3.4. Преобразование цветового пространства

Матрица, в которую загружено изображение, может содержать один канал (черно-белое изображение или в оттенках серого, тип `CV_8UC1`), три канала (формат `BGR`: `[blue, green, red]`, тип `CV_8UC3`) или четыре канала — при наличии альфа-канала (формат `BGRA`: `[blue, green, red, alpha]`, тип `CV_8UC4`). Обратите внимание на порядок следования компонентов цвета по умолчанию: `BGR`, а не `RGB`.

Диапазонами значений для компонентов `BGR` являются:

- `0...255` — для изображения `CV_8U`;
- `0...65 535` — для изображения `CV_16U`;
- `0.0...1.0` — для изображения `CV_32F`.

С помощью статического метода `cvtColor()` из класса `Imgproc` можно выполнить преобразование цветового пространства. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void cvtColor(Mat src, Mat dst, int code)
public static void cvtColor(Mat src, Mat dst, int code, int dstCn)
```

В первом параметре указывается исходное изображение, во втором — ссылка на матрицу, в которой будет сохранено новое изображение, а в третьем — константа, задающая тип преобразования. В параметре `dstCn` можно дополнительно указать количество каналов в новом изображении.

> ***ОБРАТИТЕ ВНИМАНИЕ!***
>
> Библиотека OpenCV поддерживает очень много цветовых моделей. Мы рассмотрим лишь самые основные. Полный список смотрите в документации к классу `Imgproc`.

3.4.1. Преобразование *BGR* в оттенки серого

Чтобы преобразовать изображение формата `BGR[A]` в оттенки серого, нужно указать следующие константы из класса `Imgproc`:

```
public static final int COLOR_BGR2GRAY
public static final int COLOR_BGRA2GRAY
```

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(0, 128, 255));
System.out.println(m.dump()); // [  0, 128, 255]
Mat m2 = new Mat();
Imgproc.cvtColor(m, m2, Imgproc.COLOR_BGR2GRAY);
System.out.println(m2.dump()); // [151]

Mat m3 = new Mat(1, 1, CvType.CV_8UC4, new Scalar(0, 128, 255, 255));
System.out.println(m3.dump()); // [  0, 128, 255, 255]
Mat m4 = new Mat();
Imgproc.cvtColor(m3, m4, Imgproc.COLOR_BGRA2GRAY);
System.out.println(m4.dump()); // [151]
```

Чтобы выполнить обратное преобразование, нужно указать следующие константы:

```
public static final int COLOR_GRAY2BGR
public static final int COLOR_GRAY2BGRA
```

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_8UC1, new Scalar(128));
System.out.println(m.dump()); // [128]
Mat m2 = new Mat();
Imgproc.cvtColor(m, m2, Imgproc.COLOR_GRAY2BGR);
System.out.println(m2.dump()); // [128, 128, 128]

Mat m3 = new Mat(1, 1, CvType.CV_8UC1, new Scalar(128));
System.out.println(m3.dump()); // [128]
Mat m4 = new Mat();
Imgproc.cvtColor(m3, m4, Imgproc.COLOR_GRAY2BGRA);
System.out.println(m4.dump()); // [128, 128, 128, 255]
```

Пример преобразования в оттенки серого во время загрузки из файла:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto1.jpg",
                           Imgcodecs.IMREAD_GRAYSCALE);
```

Прочие константы:

```
public static final int COLOR_RGB2GRAY
public static final int COLOR_RGBA2GRAY
public static final int COLOR_GRAY2RGB
public static final int COLOR_GRAY2RGBA
```

3.4.2. Преобразование *BGR* в *RGB*

Чтобы преобразовать изображение формата `BGR[A]` в формат `RGB[A]` и обратно, нужно указать следующие константы из класса `Imgproc`:

```
public static final int COLOR_BGR2RGB
public static final int COLOR_BGR2RGBA
public static final int COLOR_BGRA2RGB
public static final int COLOR_BGRA2RGBA
public static final int COLOR_RGB2BGR
public static final int COLOR_RGB2BGRA
public static final int COLOR_RGBA2BGR
public static final int COLOR_RGBA2BGRA
```

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(0, 128, 255));
System.out.println(m.dump()); // [  0, 128, 255]
Mat m2 = new Mat();
Imgproc.cvtColor(m, m2, Imgproc.COLOR_BGR2RGB);
System.out.println(m2.dump()); // [255, 128,   0]
```

```
Mat m3 = new Mat();
Imgproc.cvtColor(m2, m3, Imgproc.COLOR_BGR2RGB);
System.out.println(m3.dump()); // [  0, 128, 255]

Mat m4 = new Mat(1, 1, CvType.CV_8UC4, new Scalar(0, 128, 255, 255));
System.out.println(m4.dump()); // [  0, 128, 255, 255]
Mat m5 = new Mat();
Imgproc.cvtColor(m4, m5, Imgproc.COLOR_BGRA2RGBA);
System.out.println(m5.dump()); // [255, 128,   0, 255]
Mat m6 = new Mat();
Imgproc.cvtColor(m5, m6, Imgproc.COLOR_RGBA2BGRA);
System.out.println(m6.dump()); // [  0, 128, 255, 255]
```

3.4.3. Добавление или удаление альфа-канала

Чтобы добавить или удалить альфа-канал, нужно указать следующие константы из класса `Imgproc`:

```
public static final int COLOR_BGR2BGRA
public static final int COLOR_BGRA2BGR
public static final int COLOR_RGB2RGBA
public static final int COLOR_RGBA2RGB
```

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(0, 128, 255));
System.out.println(m.dump()); // [  0, 128, 255]
Mat m2 = new Mat();
Imgproc.cvtColor(m, m2, Imgproc.COLOR_BGR2BGRA);
System.out.println(m2.dump()); // [  0, 128, 255, 255]
Mat m3 = new Mat();
Imgproc.cvtColor(m2, m3, Imgproc.COLOR_BGRA2BGR);
System.out.println(m3.dump()); // [  0, 128, 255]
```

3.4.4. Преобразование *BGR* в *HSV*

Чтобы преобразовать изображение форматов `BGR` и `RGB` в формат `HSV` (`HSB`) и обратно, нужно указать следующие константы из класса `Imgproc`:

```
public static final int COLOR_BGR2HSV
public static final int COLOR_HSV2BGR
public static final int COLOR_RGB2HSV
public static final int COLOR_HSV2RGB
```

Компоненты цвета в модели `HSV` (`HSB`):

- `H` (`Hue`) — цветовой тон. Обычно значение в диапазоне `0...360`, но для изображений типа `CV_8U` это значение делится пополам, чтобы уместиться в диапазон `0...255`. В итоге получается диапазон `0...179`;

- `S` (`Saturation`) — насыщенность. Обычно значение в диапазоне `0.0...1.0`, но для изображений типа `CV_8U` диапазон `0...255`;
- `V` (`Value`) — значение цвета или яркость. Обычно значение в диапазоне `0.0...1.0`, но для изображений типа `CV_8U` диапазон `0...255`.

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(43, 0, 255));
System.out.println(m.dump()); // [ 43,   0, 255]
Mat m2 = new Mat();
Imgproc.cvtColor(m, m2, Imgproc.COLOR_BGR2HSV);
System.out.println(m2.dump()); // [175, 255, 255]
Mat m3 = new Mat();
Imgproc.cvtColor(m2, m3, Imgproc.COLOR_HSV2BGR);
System.out.println(m3.dump()); // [ 42,   0, 255]
```

Чтобы получить значения в обычном диапазоне, следует предварительно преобразовать изображение к типу `CV_32F` с помощью метода `convertTo()`:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(43, 0, 255));
Mat m_32f = new Mat();
m.convertTo(m_32f, CvType.CV_32F, 1.0 / 255);
System.out.println(m_32f.dump()); // [0.16862746, 0, 1]
Mat m2 = new Mat();
Imgproc.cvtColor(m_32f, m2, Imgproc.COLOR_BGR2HSV);
System.out.println(m2.dump());    // [349.88235, 0.99999988, 1]
```

3.4.5. Преобразование *BGR* в *HLS*

Чтобы преобразовать изображения форматов `BGR` и `RGB` в формат `HLS` и обратно, нужно указать следующие константы из класса `Imgproc`:

```
public static final int COLOR_BGR2HLS
public static final int COLOR_HLS2BGR
public static final int COLOR_RGB2HLS
public static final int COLOR_HLS2RGB
```

Компоненты цвета в модели `HLS`:

- `H` (`Hue`) — цветовой тон. Обычно значение в диапазоне `0...360`, но для изображений типа `CV_8U` это значение делится пополам, чтобы уместиться в диапазон `0...255`. В итоге получается диапазон `0...179`;
- `L` (`Lightness`) — светлота. Обычно значение в диапазоне `0.0...1.0`, но для изображений типа `CV_8U` диапазон `0...255`;
- `S` (`Saturation`) — насыщенность. Обычно значение в диапазоне `0.0...1.0`, но для изображений типа `CV_8U` диапазон `0...255`.

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(43, 0, 255));
System.out.println(m.dump()); // [ 43,   0, 255]
```

```
Mat m2 = new Mat();
Imgproc.cvtColor(m, m2, Imgproc.COLOR_BGR2HLS);
System.out.println(m2.dump()); // [175, 128, 255]
Mat m3 = new Mat();
Imgproc.cvtColor(m2, m3, Imgproc.COLOR_HLS2BGR);
System.out.println(m3.dump()); // [ 43,   1, 255]
```

Чтобы получить значения в обычном диапазоне, следует предварительно преобразовать изображение к типу `CV_32F` с помощью метода `convertTo()`:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(43, 0, 255));
Mat m_32f = new Mat();
m.convertTo(m_32f, CvType.CV_32F, 1.0 / 255);
System.out.println(m_32f.dump()); // [0.16862746, 0, 1]
Mat m2 = new Mat();
Imgproc.cvtColor(m_32f, m2, Imgproc.COLOR_BGR2HLS);
System.out.println(m2.dump());    // [349.88235, 0.5, 1]
```

3.4.6. Преобразование *BGR* в *Lab*

Чтобы преобразовать изображения форматов `BGR` и `RGB` в формат `Lab` и обратно, нужно указать следующие константы из класса `Imgproc`:

```
public static final int COLOR_BGR2Lab
public static final int COLOR_Lab2BGR
public static final int COLOR_RGB2Lab
public static final int COLOR_Lab2RGB
```

Компоненты цвета в модели `Lab`:

- `L` (`Lightness`) — светлота. Обычно значение в диапазоне `0...100`, но для изображений типа `CV_8U` в диапазоне `0...255`;
- `a` — положение цвета от зеленого до красного. Обычно значение в диапазоне значений `-127...127`, но для изображений типа `CV_8U` в диапазоне `0...255`;
- `b` — положение цвета от синего до желтого. Обычно значение в диапазоне значений `-127...127`, но для изображений типа `CV_8U` в диапазоне `0...255`.

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(43, 0, 255));
System.out.println(m.dump()); // [ 43,   0, 255]
Mat m2 = new Mat();
Imgproc.cvtColor(m, m2, Imgproc.COLOR_BGR2Lab);
System.out.println(m2.dump()); // [136, 209, 180]
Mat m3 = new Mat();
Imgproc.cvtColor(m2, m3, Imgproc.COLOR_Lab2BGR);
System.out.println(m3.dump()); // [ 43,   0, 255]
```

Чтобы получить значения в обычном диапазоне, следует предварительно преобразовать изображение к типу `CV_32F` с помощью метода `convertTo()`:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(43, 0, 255));
Mat m_32f = new Mat();
m.convertTo(m_32f, CvType.CV_32F, 1.0 / 255);
System.out.println(m_32f.dump()); // [0.16862746, 0, 1]
Mat m2 = new Mat();
Imgproc.cvtColor(m_32f, m2, Imgproc.COLOR_BGR2Lab);
System.out.println(m2.dump());    // [53.430176, 80.609375, 52.03125]
```

3.5. Изменение типа изображения

Как вы уже знаете, для хранения изображений в основном используются типы матриц `CV_8U` (в диапазоне значений от `0` до `255`), `CV_16U` (в диапазоне значений от `0` до `65535`) и `CV_32F` (в диапазоне значений от `0.0` до `1.0`). Давайте научимся преобразовывать один тип изображения в другой.

3.5.1. Преобразование типа *CV_8U* в *CV_32F*

Чтобы преобразовать матрицу типа `CV_8U` в `CV_32F` (в диапазоне значений от `0.0` до `1.0`) и обратно, нужно воспользоваться методом `convertTo()` из класса `Mat`. Форматы метода:

```
public void convertTo(Mat m, int rtype)
public void convertTo(Mat m, int rtype, double alpha)
public void convertTo(Mat m, int rtype, double alpha, double beta)
```

В первом параметре указываем ссылку на матрицу, в которую будут скопированы данные, а во втором параметре — новый тип. Значение параметра `alpha` будет умножено на значение компонента цвета, а значение `beta` — прибавлено.

Пример преобразования:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(43, 0, 255));
System.out.println(m.dump()); // [ 43,   0, 255]
Mat m2 = new Mat();
m.convertTo(m2, CvType.CV_32F, 1.0 / 255);
System.out.println(m2.dump()); // [0.16862746, 0, 1]
Mat m3 = new Mat();
m2.convertTo(m3, CvType.CV_8U, 255);
System.out.println(m3.dump()); // [ 43,   0, 255]
```

3.5.2. Преобразование типа *CV_16U* в *CV_32F*

Пример преобразования матрицы типа `CV_16U` (в диапазоне значений от `0` до `65 535`) в `CV_32F` (в диапазоне значений от `0.0` до `1.0`) и обратно:

```
Mat m = new Mat(1, 1, CvType.CV_16UC3, new Scalar(0, 32768, 65535));
System.out.println(m.dump()); // [0, 32768, 65535]
Mat m2 = new Mat();
m.convertTo(m2, CvType.CV_32F, 1.0 / 65535);
```

```
System.out.println(m2.dump()); // [0, 0.50000763, 1]
Mat m3 = new Mat();
m2.convertTo(m3, CvType.CV_16U, 65535);
System.out.println(m3.dump()); // [0, 32768, 65535]
```

3.5.3. Преобразование типа *CV_8U* в *CV_16U*

Пример преобразования матрицы типа `CV_8U` в `CV_16U` (в диапазоне значений от `0` до `65535`) и обратно:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(0, 128, 255));
System.out.println(m.dump()); // [  0, 128, 255]
Mat m2 = new Mat();
m.convertTo(m2, CvType.CV_16U, 65535.0 / 255);
System.out.println(m2.dump()); // [0, 32896, 65535]
Mat m3 = new Mat();
m2.convertTo(m3, CvType.CV_8U, 255.0 / 65535);
System.out.println(m3.dump()); // [  0, 128, 255]
```

3.6. Преобразование *Mat* в *BufferedImage*

Чтобы иметь возможность отобразить изображение на экране в приложении Swing или работать с изображением средствами стандартной библиотеки языка Java, нужно уметь преобразовывать объект класса `Mat` в объект класса `BufferedImage`. К сожалению, стандартного метода для преобразования нет, поэтому приходится экспериментировать и искать быстрые способы преобразования. Если попытаться просто выполнить посимвольное копирование, то можно обнаружить, что такой метод работает непозволительно долго. После множества проб и ошибок получился метод `MatToBufferedImage()` (листинг 3.1), который работает не слишком медленно. Именно его мы будем использовать в дальнейших примерах этой книги. Возможно, вы сможете создать более быстрый способ преобразования.

Листинг 3.1. Преобразование `Mat` в `BufferedImage`

```
public static BufferedImage MatToBufferedImage(Mat m) {
   if (m == null || m.empty()) return null;
   if (m.depth() == CvType.CV_8U) {}
   else if (m.depth() == CvType.CV_16U) { // CV_16U => CV_8U
      Mat m_16 = new Mat();
      m.convertTo(m_16, CvType.CV_8U, 255.0 / 65535);
      m = m_16;
   }
   else if (m.depth() == CvType.CV_32F) { // CV_32F => CV_8U
      Mat m_32 = new Mat();
      m.convertTo(m_32, CvType.CV_8U, 255);
      m = m_32;
   }
```

```
   else
      return null;
   int type = 0;
   if (m.channels() == 1)
      type = BufferedImage.TYPE_BYTE_GRAY;
   else if (m.channels() == 3)
      type = BufferedImage.TYPE_3BYTE_BGR;
   else if (m.channels() == 4)
      type = BufferedImage.TYPE_4BYTE_ABGR;
   else
      return null;

   byte[] buf = new byte[m.channels() * m.cols() * m.rows()];
   m.get(0, 0, buf);
   byte tmp = 0;
   if (m.channels() == 4) { // BGRA => ABGR
      for (int i = 0; i < buf.length; i += 4) {
         tmp = buf[i + 3];
         buf[i + 3] = buf[i + 2];
         buf[i + 2] = buf[i + 1];
         buf[i + 1] = buf[i];
         buf[i] = tmp;
      }
   }

   BufferedImage image = new BufferedImage(m.cols(), m.rows(), type);
   byte[] data =
      ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
   System.arraycopy(buf, 0, data, 0, buf.length);
   return image;
}
```

Для преобразования объекта класса `BufferedImage` в объект класса `Mat` также нет стандартного метода. Поэтому напишем метод `BufferedImageToMat()` самостоятельно (листинг 3.2).

Листинг 3.2. Преобразование `BufferedImage` в `Mat`

```
public static Mat BufferedImageToMat(BufferedImage img) {
   if (img == null) return new Mat();
   int type = 0;
   if (img.getType() == BufferedImage.TYPE_BYTE_GRAY) {
      type = CvType.CV_8UC1;
   }
   else if (img.getType() == BufferedImage.TYPE_3BYTE_BGR) {
      type = CvType.CV_8UC3;
   }
```

```
    else if (img.getType() == BufferedImage.TYPE_4BYTE_ABGR) {
       type = CvType.CV_8UC4;
    }
    else return new Mat();

    Mat m = new Mat(img.getHeight(), img.getWidth(), type);
    byte[] data =
          ((DataBufferByte) img.getRaster().getDataBuffer()).getData();
    if (type == CvType.CV_8UC1 || type == CvType.CV_8UC3) {
       m.put(0, 0, data);
       return m;
    }
    byte[] buf = Arrays.copyOf(data, data.length);
    byte tmp = 0;
    for (int i = 0; i < buf.length; i += 4) { // ABGR => BGRA
       tmp = buf[i];
       buf[i] = buf[i + 1];
       buf[i + 1] = buf[i + 2];
       buf[i + 2] = buf[i + 3];
       buf[i + 3] = tmp;
    }
    m.put(0, 0, buf);
    return m;
}
```

3.7. Преобразование *Mat* в *WritableImage*

На смену Swing приходит великолепная библиотека JavaFX. Чтобы иметь возможность отображать изображение в этой библиотеке, напишем метод преобразования объекта класса `Mat` в объект класса `WritableImage` и будем его использовать в дальнейших примерах.

Разработчики библиотеки JavaFX предусмотрели возможность преобразования объекта класса `Image` (`WritableImage`) в объект класса `BufferedImage`, а также возможность выполнять обратную операцию. Для этого предназначены следующие статические методы из класса `SwingFXUtils`:

```
import javafx.embed.swing.SwingFXUtils;
import java.awt.image.BufferedImage;
```

- `fromFXImage()` — преобразует объект класса `Image` (`WritableImage`) в объект класса `BufferedImage`. Возвращает объект изображения или значение `null`. Формат метода:

  ```
  public static BufferedImage fromFXImage(Image img, BufferedImage bimg)
  ```

  Пример:

  ```
  WritableImage wim = new WritableImage(200, 200);
  // Что-то рисуем на изображении (фрагмент опущен)
  BufferedImage bim = SwingFXUtils.fromFXImage(wim, null);
  ```

- `toFXImage()` — преобразует объект класса `BufferedImage` в объект класса `WritableImage`. Формат метода:

```
public static WritableImage toFXImage(BufferedImage bimg,
                                      WritableImage wimg)
```

Пример обратного преобразования:

```
WritableImage wim2 = SwingFXUtils.toFXImage(bim, null);
```

Используя метод `toFXImage()` совместно с методом `MatToBufferedImage()`, который мы создали в предыдущем разделе (см. листинг 3.1), можно очень просто преобразовать объект класса `Mat` в объект класса `WritableImage` (листинг 3.3).

Листинг 3.3. Преобразование `Mat` в `WritableImage`

```
public static WritableImage MatToWritableImage(Mat m) {
   BufferedImage bim = CvUtils.MatToBufferedImage(m);
   if (bim == null) return null;
   else return SwingFXUtils.toFXImage(bim, null);
}
```

В этом случае создается промежуточный объект `BufferedImage`. Давайте попробуем избавиться от промежуточного объекта и немного ускорим код (листинг 3.4).

Листинг 3.4. Преобразование `Mat` в `WritableImage` (ускоренный вариант)

```
public static WritableImage MatToImageFX(Mat m) {
   if (m == null || m.empty()) return null;
   if (m.depth() == CvType.CV_8U) {}
   else if (m.depth() == CvType.CV_16U) {
      Mat m_16 = new Mat();
      m.convertTo(m_16, CvType.CV_8U, 255.0 / 65535);
      m = m_16;
   }
   else if (m.depth() == CvType.CV_32F) {
      Mat m_32 = new Mat();
      m.convertTo(m_32, CvType.CV_8U, 255);
      m = m_32;
   }
   else
      return null;

   if (m.channels() == 1) {
      Mat m_bgra = new Mat();
      Imgproc.cvtColor(m, m_bgra, Imgproc.COLOR_GRAY2BGRA);
      m = m_bgra;
   }
```

```
   else if (m.channels() == 3) {
      Mat m_bgra = new Mat();
      Imgproc.cvtColor(m, m_bgra, Imgproc.COLOR_BGR2BGRA);
      m = m_bgra;
   }
   else if (m.channels() == 4) { }
   else
      return null;

   byte[] buf = new byte[m.channels() * m.cols() * m.rows()];
   m.get(0, 0, buf);

   WritableImage wim = new WritableImage(m.cols(), m.rows());
   PixelWriter pw = wim.getPixelWriter();
   pw.setPixels(0, 0, m.cols(), m.rows(),
                WritablePixelFormat.getByteBgraInstance(),
                buf, 0, m.cols() * 4);
   return wim;
}
```

При использовании матрицы типа `CV_8UC4` метод `MatToImageFX()` будет работать быстрее метода `MatToWritableImage()`, т. к. не потребуется выполнять никаких лишних преобразований. Если же тип другой, то время, затраченное на преобразование типа, может быть сопоставимо со временем создания объекта класса `BufferedImage`. В любом случае, преобразование `Mat` в объект класса `BufferedImage` выполняется быстрее, чем в объект класса `WritableImage`.

Теперь создадим метод, позволяющий преобразовать объекты классов `Image` и `WritableImage` в объект класса `Mat` (листинг 3.5).

Листинг 3.5. Преобразование `Image` (`WritableImage`) в `Mat`

```
public static Mat ImageFXToMat(javafx.scene.image.Image img) {
   if (img == null) return new Mat();
   PixelReader pr = img.getPixelReader();
   int w = (int) img.getWidth();
   int h = (int) img.getHeight();
   byte[] buf = new byte[4 * w * h];
   pr.getPixels(0, 0, w, h, WritablePixelFormat.getByteBgraInstance(),
                buf, 0, w * 4);
   Mat m = new Mat(h, w, CvType.CV_8UC4);
   m.put(0, 0, buf);
   return m;
}
```

3.8. Сохранение матрицы в бинарный файл

При сохранении изображения в некоторых форматах возможна потеря данных — например, при сохранении в формате JPEG. Конечно, мы можем выбрать формат, в котором используется сжатие без потерь, — например, PNG, но при сохранении файла потеряем время на сжатие данных, а при загрузке — на распаковку файла. Чтобы не терять качество и время на сжатие, давайте напишем метод `saveMat()`, позволяющий сохранить объект класса `Mat` с изображением в бинарный файл с расширением mat без сжатия (листинг 3.6). Данные будем хранить в формате `CV_8U` с последовательностью цветовых компонентов `Gray`, `BGR` или `BGRA`, в зависимости от типа матрицы. Если матрица имеет тип `CV_16U` или `CV_32F`, то выполним преобразование в тип `CV_8U`. Если сохранить не удалось, вернем значение `false`.

Листинг 3.6. Сохранение `Mat` в бинарный файл

```
public static boolean saveMat(Mat m, String path) {
   if (m == null || m.empty()) return false;
   if (path == null || path.length() < 5 || !path.endsWith(".mat"))
      return false;
   if (m.depth() == CvType.CV_8U) {}
   else if (m.depth() == CvType.CV_16U) {
      Mat m_16 = new Mat();
      m.convertTo(m_16, CvType.CV_8U, 255.0 / 65535);
      m = m_16;
   }
   else if (m.depth() == CvType.CV_32F) {
      Mat m_32 = new Mat();
      m.convertTo(m_32, CvType.CV_8U, 255);
      m = m_32;
   }
   else
      return false;

   if (m.channels() == 2 || m.channels() > 4) return false;

   byte[] buf = new byte[m.channels() * m.cols() * m.rows()];
   m.get(0, 0, buf);

   try (
      OutputStream out = new FileOutputStream(path);
      BufferedOutputStream bout = new BufferedOutputStream(out);
      DataOutputStream dout = new DataOutputStream(bout);
   )
   {
      dout.writeInt(m.rows());
      dout.writeInt(m.cols());
```

```
      dout.writeInt(m.channels());
      dout.write(buf);
      dout.flush();
   } catch (Exception e) {
      return false;
   }
   return true;
}
```

Теперь напишем метод `loadMat()`, позволяющий загрузить матрицу из бинарного файла (листинг 3.7). Если загрузить не удалось, вернем пустую матрицу.

Листинг 3.7. Загрузка `Mat` из бинарного файла

```
public static Mat loadMat(String path) {
   if (path == null || path.length() < 5 || !path.endsWith(".mat"))
      return new Mat();
   File f = new File(path);
   if (!f.exists() || !f.isFile()) return new Mat();
   try (
      InputStream in = new FileInputStream(path);
      BufferedInputStream bin = new BufferedInputStream(in);
      DataInputStream din = new DataInputStream(bin);
   )
   {
      int rows = din.readInt();
      if (rows < 1) return new Mat();
      int cols = din.readInt();
      if (cols < 1) return new Mat();
      int ch = din.readInt();
      int type = 0;
      if (ch == 1) {
         type = CvType.CV_8UC1;
      }
      else if (ch == 3) {
         type = CvType.CV_8UC3;
      }
      else if (ch == 4) {
         type = CvType.CV_8UC4;
      }
      else return new Mat();

      int size = ch * cols * rows;
      byte[] buf = new byte[size];
      int rsize = din.read(buf);
      if (size != rsize) return new Mat();
```

```
      Mat m = new Mat(rows, cols, type);
      m.put(0, 0, buf);
      return m;
   } catch (Exception e) { }
   return new Mat();
}
```

3.9. Класс *CvUtils* и шаблон приложения JavaFX

Все созданные нами в этой главе методы вынесем в классы `CvUtils` (листинг 3.8) и `CvUtilsFX` (листинг 3.9), а также добавим методы для просмотра изображения в окне в приложениях Swing и JavaFX.

Листинг 3.8. Класс `CvUtils`

```
package application;

import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;
import java.io.*;
import java.util.Arrays;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;

import org.opencv.core.CvType;
import org.opencv.core.Mat;

public class CvUtils {

   public static BufferedImage MatToBufferedImage(Mat m) {
      // Листинг 3.1
   }

   public static Mat BufferedImageToMat(BufferedImage img) {
      // Листинг 3.2
   }

   public static boolean saveMat(Mat m, String path) {
      // Листинг 3.6
   }
```

```
   public static Mat loadMat(String path) {
      // Листинг 3.7
   }

   public static void showImage(Mat img, String title) {
      BufferedImage im = MatToBufferedImage(img);
      if (im == null) return;

      int w = 1000, h = 600;
      JFrame window = new JFrame(title);
      window.setSize(w, h);
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      ImageIcon imageIcon = new ImageIcon(im);
      JLabel label = new JLabel(imageIcon);

      JScrollPane pane = new JScrollPane(label);
      window.setContentPane(pane);

      if (im.getWidth() < w && im.getHeight() < h) {
         window.pack();
      }
      window.setLocationRelativeTo(null);
      window.setVisible(true);
   }
}
```

Пример загрузки и просмотра изображения в приложении Swing:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtils.showImage(img, "Текст в заголовке окна");
```

Листинг 3.9. Класс `CvUtilsFX`

```
package application;

import java.awt.image.BufferedImage;

import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.imgproc.Imgproc;

import javafx.embed.swing.SwingFXUtils;
import javafx.scene.Scene;
```

```
import javafx.scene.control.ScrollPane;
import javafx.scene.image.*;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class CvUtilsFX {

   public static WritableImage MatToWritableImage(Mat m) {
      // Листинг 3.3
   }

   public static WritableImage MatToImageFX(Mat m) {
      // Листинг 3.4
   }

   public static Mat ImageFXToMat(Image img) {
      // Листинг 3.5
   }

   public static void showImage(Mat img, String title) {
      Image im = MatToImageFX(img);

      Stage window = new Stage();
      ScrollPane sp = new ScrollPane();
      ImageView iv = new ImageView();
      if (im != null) {
         iv.setImage(im);
         if (im.getWidth() < 1000) {
            sp.setPrefWidth(im.getWidth() + 5);
         }
         else sp.setPrefWidth(1000.0);
         if (im.getHeight() < 700) {
            sp.setPrefHeight(im.getHeight() + 5);
         }
         else sp.setPrefHeight(700.0);
      }
      sp.setContent(iv);
      sp.setPannable(true);

      BorderPane box = new BorderPane();
      box.setCenter(sp);

      Scene scene = new Scene(box);
      window.setScene(scene);
      window.setTitle(title);
      window.show();
   }
}
```

Обратите внимание: вызывать метод `showImage()` из класса `CvUtilsFX` можно только из приложения JavaFX, т. е. из потока с названием `JavaFX Application Thread`, в противном случае возникнет исключение. Давайте создадим шаблон приложения JavaFX (листинг 3.10).

Листинг 3.10. Шаблон приложения JavaFX

```
package application;

import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.imgcodecs.Imgcodecs; // 3.3
// import org.opencv.highgui.Highgui;  // 2.4

import javafx.application.Application;
import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

public class Main extends Application {

   static {
      System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
   }

   public static void main(String[] args) {
      Application.launch(args);
   }

   public void start(Stage stage) throws Exception {
      VBox root = new VBox(15.0);
      root.setAlignment(Pos.CENTER);

      Button button = new Button("Выполнить");
      button.setOnAction(this::onClickButton);
      root.getChildren().add(button);

      Scene scene = new Scene(root, 400.0, 150.0);
      stage.setTitle("OpenCV " + Core.VERSION);
      stage.setScene(scene);
      stage.setOnCloseRequest(event -> {
         Platform.exit();
      });
```

```
        stage.show();
    }

    private void onClickButton(ActionEvent e) {
        // Загружаем изображение из файла
        Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
        // 2.4
        // Mat img = Highgui.imread("C:\\book\\opencv\\foto.jpg");
        if (img.empty()) {
            System.out.println("Не удалось загрузить изображение");
            return;
        }
        // Обрабатываем изображение
        // Отображаем в отдельном окне
        CvUtilsFX.showImage(img, "Текст в заголовке окна");
    }
}
```

Итак, загрузку, обработку и отображение объекта `Mat` мы будем производить внутри обработчика нажатия кнопки — внутри метода `onClickButton()`. В дальнейших примерах, в целях экономии места в книге, код шаблона больше приводиться не будет. По умолчанию подразумевается, что код примера должен быть вставлен внутрь обработчика нажатия кнопки из этого шаблона. Исключением является код примеров для работы с видеофайлами и видеокамерами.

3.10. Чтение кадров из видеофайла

Для работы с видеофайлами и видеокамерами в библиотеке OpenCV предназначен класс `VideoCapture`. Инструкция импорта в версии 3.3:

```
import org.opencv.videoio.VideoCapture;
```

В версии 2.4 класс `VideoCapture` расположен в другом пакете:

```
import org.opencv.highgui.VideoCapture;
```

Конструкторы класса:

```
VideoCapture()
VideoCapture(String filename)
VideoCapture(String filename, int apiPreference) // Нет в версии 2.4
```

Первый конструктор создает объект с настройками по умолчанию, а второй — позволяет указать путь к видеофайлу (или URL видеопотока). Обратите внимание: в пути не должно быть русских букв. В параметре `apiPreference` можно указать следующие константы из класса `Videoio`:

```
import org.opencv.videoio.Videoio;
public static final int CAP_FFMPEG
```

```
public static final int CAP_IMAGES
public static final int CAP_DSHOW
```

Выведем значения констант:

```
System.out.println(Videoio.CAP_FFMPEG); // 1900
System.out.println(Videoio.CAP_IMAGES); // 2000
System.out.println(Videoio.CAP_DSHOW);  // 700
```

Если видеофайл не удается прочитать, то нужно скопировать библиотеку opencv_ffmpeg330_64.dll (в версии 2.4 opencv_ffmpeg2413_64.dll) из папки C:\opencv_3_3\build\x64\vc14\bin (в версии 2.4.13 C:\opencv_2_4\build\x64\vc12\bin) в папку с проектом. После этого станут доступны все форматы, поддерживаемые FFmpeg, а их довольно много. Пример:

```
VideoCapture capture = new VideoCapture(
                           "C:\\book\\opencv\\video1.avi");
if (!capture.isOpened()) {
   System.out.println("Не удалось открыть видео");
   return;
}
```

В этом примере мы воспользовались методом `isOpened()`, который возвращает значение `true`, если файл удалось прочитать, и `false` — в противном случае. Формат метода:

```
public boolean isOpened()
```

Вместо указания пути к видеофайлу можно задать путь к последовательности кадров, сохраненных в различных форматах, — например, в JPEG. Все файлы этой последовательности должны быть пронумерованы. В составе пути указывается формат названия файлов — например, `%02d.jpg`. Цифра `2` задает количество чисел в названии файла, а `0` — говорит, что перед номером кадра указана цифра `0`, если номер меньше `10`. Этому формату соответствуют названия `01.jpg`, `02.jpg`, ..., `99.jpg`.

Пример:

```
VideoCapture capture = new VideoCapture();
if (!capture.open("C:\\book\\opencv\\sequence\\%02d.jpg")) {
   System.out.println("Не удалось открыть");
   return;
}
```

В этом примере мы воспользовались методом `open()`, который открывает видеофайл и возвращает значение `true`, если операция успешно выполнена, и `false` — если видеофайл открыть не удалось. Форматы метода:

```
public boolean open(String filename)
public boolean open(String filename, int apiPreference) // Нет в 2.4
```

При работе с видео в режиме реального времени важна скорость обработки и отображения кадров. Так как преобразование объекта класса `Mat` в объект класса `BufferedImage` выполняется быстрее, чем в объект класса `WritableImage`, для отображения кадров видео воспользуемся приложением Swing (листинг 3.11).

Листинг 3.11. Чтение кадров из видеофайла

```
package application;

import java.awt.image.BufferedImage;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;
import org.opencv.videoio.VideoCapture;

public class Video {

   static {
      System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
   }

   public static void main(String[] args) {
      JFrame window = new JFrame("Просмотр видео");
      window.setSize(1000, 600);
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      window.setLocationRelativeTo(null);
      JLabel label = new JLabel();
      window.setContentPane(label);
      window.setVisible(true);

      VideoCapture capture = new VideoCapture(
                                 "C:\\book\\opencv\\video1.avi");
      if (!capture.isOpened()) {
         System.out.println("Не удалось открыть видео");
         return;
      }
      Mat frame = new Mat();
      BufferedImage img = null;
      while (capture.read(frame)) {
         Imgproc.resize(frame, frame, new Size(960, 540));
         // Здесь можно вставить код обработки кадра
         img = CvUtils.MatToBufferedImage(frame);
         if (img != null) {
            ImageIcon imageIcon = new ImageIcon(img);
            label.setIcon(imageIcon);
            label.repaint();
            window.pack();
         }
```

```
            try {
               Thread.sleep(10);
            } catch (InterruptedException e) {}
         }
         System.out.println("Выход");
         capture.release();
      }
}
```

3.11. Захват кадров с веб-камеры

Библиотека OpenCV позволяет не только читать кадры из видеофайла, но и захватывать кадры с внешней камеры (например, с веб-камеры) в режиме реального времени. Для подключения к камере используется следующий формат конструктора класса `VideoCapture`:

```
VideoCapture(int index)
```

В качестве значения параметра `index` указывается индекс камеры в системе. Если камера одна, то следует указать значение `0`. Проверить успешность подключения позволяет метод `isOpened()`.

Пример:

```
VideoCapture camera = new VideoCapture(0);
if (!camera.isOpened()) {
   System.out.println("Не удалось подключиться к камере");
   return;
}
```

Для подключения к камере можно также воспользоваться методом `open()`. Форматы метода:

```
public boolean open(int index)
public boolean open(int cameraNum, int apiPreference) // Нет в 2.4
```

Метод `open()` сначала вызывает метод `release()` для отключения соединения, затем подключается к камере с указанным индексом и возвращает значение `true` при успешном подключении.

Пример:

```
VideoCapture camera = new VideoCapture();
if (!camera.open(0)) {
   System.out.println("Не удалось подключиться к камере");
   return;
}
```

Для отключения от камеры используется метод `release()`. Формат метода:

```
public void release()
```

Захватить кадр с камеры позволяет метод `grab()`. Если кадр успешно захвачен, то метод возвращает значение `true`. Формат метода:

```
public boolean grab()
```

После успешного захвата следует вызвать метод `retrieve()` для обработки захваченного кадра и записи его в матрицу `image`. Метод `retrieve()` позволяет также получить кадр с камеры, имеющей несколько каналов, — например, со стереокамеры. Если операция успешно выполнена, метод вернет значение `true`. Форматы метода:

```
public boolean retrieve(Mat image)
public boolean retrieve(Mat image, int flag)
```

Пример:

```
Mat frame = new Mat();
BufferedImage img = null;
if (camera.grab()) {
   if (camera.retrieve(frame)) {
      img = CvUtils.MatToBufferedImage(frame);
      if (img != null) {
         ImageIcon imageIcon = new ImageIcon(img);
         label.setIcon(imageIcon);
         label.repaint();
      }
   }
   else {
      System.out.println("Не удалось обработать кадр");
   }
}
else {
   System.out.println("Не удалось захватить кадр");
}
```

Выполнить сразу и захват кадра и его обработку позволяет метод `read()`. Если операция успешно выполнена, метод вернет значение `true`. Формат метода:

```
public boolean read(Mat image)
```

Получить информацию о свойствах камеры позволяет метод `get()`. Формат метода:

```
public double get(int propId)
```

Полный список констант, которые можно указать в параметре `propId`, смотрите в документации.

Например, получим ширину и высоту кадра:

```
System.out.println(camera.get(Videoio.CAP_PROP_FRAME_WIDTH));
System.out.println(camera.get(Videoio.CAP_PROP_FRAME_HEIGHT));
```

Пример для версии 2.4:

```
System.out.println(camera.get(Highgui.CV_CAP_PROP_FRAME_WIDTH));
System.out.println(camera.get(Highgui.CV_CAP_PROP_FRAME_HEIGHT));
```

Задать значения можно с помощью метода `set()`. Формат метода:

```
public boolean set(int propId, double value)
```

Например, указать предпочтительные размеры кадра можно так:

```
camera.set(Videoio.CAP_PROP_FRAME_WIDTH, 640);
camera.set(Videoio.CAP_PROP_FRAME_HEIGHT, 480);
```

Пример для версии 2.4:

```
camera.set(Highgui.CV_CAP_PROP_FRAME_WIDTH, 640);
camera.set(Highgui.CV_CAP_PROP_FRAME_HEIGHT, 480);
```

При работе с камерой важно отключиться от нее до завершения работы приложения. В противном случае поток продолжит работу даже после вызова инструкции `System.exit(0)`. Поэтому создадим обработчик, который будет вызываться при нажатии кнопки **Закрыть** в заголовке окна, и внутри него будем проверять статус подключения к камере. Кроме того, напишем обработчик нажатия клавиши <Esc>, внутри которого изменим значение флага. Получать кадры мы будем в бесконечном цикле, на каждой итерации проверяя значение флага. Если флаг имеет значение `false` (пользователь нажал клавишу <Esc>), то выйдем из цикла, отключившись от камеры и изменив перед этим статус подключения. Полный код захвата кадров с веб-камеры и отображения в окне показан в листинге 3.12.

Листинг 3.12. Захват кадров с веб-камеры

```
package application;

import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.BufferedImage;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.videoio.VideoCapture;
import org.opencv.videoio.Videoio;

public class Camera {

   static {
      System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
   }
```

```
public static boolean isRun = true;
public static boolean isEnd = false;

public static void main(String[] args) {

   JFrame window = new JFrame(
                       "Нажмите <Esc> для отключения от камеры");
   window.setSize(640, 480);
   window.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
   window.setLocationRelativeTo(null);
   // Обработка нажатия кнопки Закрыть в заголовке окна
   window.addWindowListener(new WindowAdapter() {
      @Override
      public void windowClosing(WindowEvent e) {
         isRun = false;
         if ( isEnd ) {
            window.dispose();
            System.exit(0);
         }
         else {
            System.out.println(
                  "Сначала нажмите <Esc>, потом Закрыть");
         }
      }
   });
   // Обработка нажатия клавиши <Esc>
   window.addKeyListener(new KeyAdapter() {
      @Override
      public void keyPressed(KeyEvent e) {
         if (e.getKeyCode() == 27) {
            isRun = false;
         }
      }
   });

   JLabel label = new JLabel();
   window.setContentPane(label);
   window.setVisible(true);
   // Подключаемся к камере
   VideoCapture camera = new VideoCapture(0);
   if (!camera.isOpened()) {
      window.setTitle("Не удалось подключиться к камере");
      isRun = false;
      isEnd = true;
      return;
   }
```

```
      try {
         // Задаем размеры кадра
         camera.set(Videoio.CAP_PROP_FRAME_WIDTH, 640);
         camera.set(Videoio.CAP_PROP_FRAME_HEIGHT, 480);
         // Считываем кадры
         Mat frame = new Mat();
         BufferedImage img = null;
         while ( isRun ) {
            if (camera.read(frame)) {
               // Здесь можно вставить код обработки кадра
               img = CvUtils.MatToBufferedImage(frame);
               if (img != null) {
                  ImageIcon imageIcon = new ImageIcon(img);
                  label.setIcon(imageIcon);
                  label.repaint();
                  window.pack();
               }
               try {
                  Thread.sleep(100); // 10 кадров в секунду
               } catch (InterruptedException e) {}
            }
            else {
               System.out.println("Не удалось захватить кадр");
               break;
            }
         }
      }
      finally {
         camera.release();
         isRun = false;
         isEnd = true;
      }
      window.setTitle("Камера отключена");
   }
}
```

Чтобы выйти из приложения, нужно сначала нажать клавишу <Esc> для отключения от камеры, а затем кнопку **Закрыть**, расположенную в заголовке окна. Можно также два раза нажать кнопку **Закрыть**.

Обратите внимание на инструкцию:

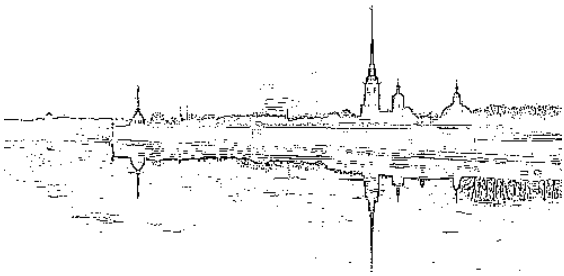
```
Thread.sleep(100); // 10 кадров в секунду
```

Изменяя значение параметра метода `sleep()`, можно регулировать количество кадров, получаемых с камеры в секунду. Если указать значение `100`, то будет около `10` кадров в секунду, а если `33` — то около `30` кадров в секунду. Если обработка кад-

ра занимает длительное время, то количество кадров в секунду, конечно же, будет другим, т. к. это не таймер, а всего лишь задержка перед повторным запросом к камере. Не убирайте эту инструкцию из кода, иначе обращений к камере будет очень много, и можно нарушить работоспособность камеры (особенно это относится к встроенным веб-камерам).

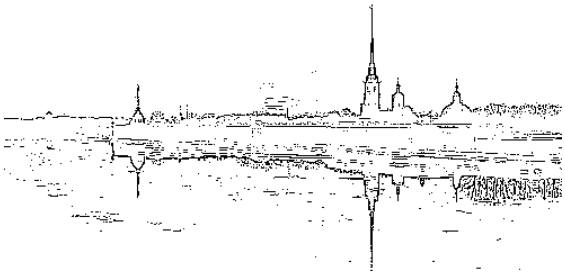
> ***Примечание***
>
> В версии 3.3 существует также возможность записи видео в файл. Для этого предназначен класс `org.opencv.videoio.VideoWriter`. Описание класса смотрите в документации.

ЧАСТЬ II



Анализ и обработка изображения

ГЛАВА 4



Рисование фигур и вывод текста на изображение

Библиотека OpenCV позволяет рисовать различные геометрические фигуры: линии, прямоугольники, эллипсы и др., а также выводить на изображение текст. Возможности довольно скромные, т. к. основная цель — просто выделить найденный объект на изображении и подписать его. Если нужно нарисовать что-то более качественно, то лучше воспользоваться возможностями стандартной библиотеки языка Java.

4.1. Указание цвета

Цвет обводки или заливки задается с помощью объекта класса `Scalar` в формате `BGR` или `Gray` (для изображений в градациях серого). Пример указания красного цвета:

```
System.out.println(new Scalar(0, 0, 255));
// [0.0, 0.0, 255.0, 0.0]
```

Более привычно было бы указывать цвет в формате `RGB` или в виде константы. Давайте добавим в наш класс `CvUtils` (см. листинг 3.8) несколько констант и методов, которыми мы будем пользоваться в дальнейших примерах (листинг 4.1).

Листинг 4.1. Указание цвета

```
public static final Scalar COLOR_BLACK = colorRGB(0, 0, 0);
public static final Scalar COLOR_WHITE = colorRGB(255, 255, 255);
public static final Scalar COLOR_RED = colorRGB(255, 0, 0);
public static final Scalar COLOR_BLUE = colorRGB(0, 0, 255);
public static final Scalar COLOR_GREEN = colorRGB(0, 128, 0);
public static final Scalar COLOR_YELLOW = colorRGB(255, 255, 0);
public static final Scalar COLOR_GRAY = colorRGB(128, 128, 128);

public static Scalar colorRGB(double red, double green, double blue) {
   return new Scalar(blue, green, red);
}
```

```
public static Scalar colorRGB(java.awt.Color c) {
   return new Scalar(c.getBlue(), c.getGreen(), c.getRed());
}

public static Scalar colorRGBA(double red, double green, double blue,
                               double alpha) {
   return new Scalar(blue, green, red, alpha);
}

public static Scalar colorRGBA(java.awt.Color c) {
   return new Scalar(c.getBlue(), c.getGreen(),
                     c.getRed(), c.getAlpha());
}
```

В класс `CvUtilsFX` (см. листинг 3.9) добавим следующие методы:

```
public static Scalar colorRGB(javafx.scene.paint.Color c) {
   return new Scalar((double) Math.round(c.getBlue() * 255),
                     (double) Math.round(c.getGreen() * 255),
                     (double) Math.round(c.getRed() * 255));
}

public static Scalar colorRGBA(javafx.scene.paint.Color c) {
   return new Scalar((double) Math.round(c.getBlue() * 255),
                     (double) Math.round(c.getGreen() * 255),
                     (double) Math.round(c.getRed() * 255),
                     (double) Math.round(c.getOpacity() * 255));
}
```

Пример указания красного цвета разными способами:

```
System.out.println(CvUtils.COLOR_RED);
// [0.0, 0.0, 255.0, 0.0]
System.out.println(CvUtils.colorRGB(255, 0, 0));
// [0.0, 0.0, 255.0, 0.0]
System.out.println(CvUtilsFX.colorRGB(javafx.scene.paint.Color.RED));
// [0.0, 0.0, 255.0, 0.0]
System.out.println(CvUtils.colorRGB(java.awt.Color.red));
// [0.0, 0.0, 255.0, 0.0]
System.out.println(CvUtilsFX.colorRGBA(javafx.scene.paint.Color.RED));
// [0.0, 0.0, 255.0, 255.0]
System.out.println(CvUtils.colorRGBA(java.awt.Color.red));
// [0.0, 0.0, 255.0, 255.0]
```

4.2. Рисование линии

Нарисовать линию позволяет статический метод `line()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void line(Mat img, Point pt1, Point pt2, Scalar color)
public static void line(Mat img, Point pt1, Point pt2,
                        Scalar color, int thickness)
public static void line(Mat img, Point pt1, Point pt2,
                        Scalar color, int thickness, int type, int shift)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

В первом параметре указывается ссылка на изображение, во втором — координаты начальной точки, в третьем — координаты конечной точки, а в четвертом — цвет в формате `BGR` или `Gray` (для изображений в градациях серого). Если координаты выходят за пределы изображения, то линия обрезается. По умолчанию линия рисуется толщиной в один пиксел. С помощью параметра `thickness` можно указать другую толщину. Если линия толстая, то она рисуется со скруглением концов. В параметре `type` указывается тип линии, а в параметре `shift` — сдвиг (значение по умолчанию: `0`).

В параметре `type` можно указать следующие константы из класса `Imgproc`:

```
public static final int LINE_4
public static final int LINE_8
public static final int LINE_AA
```

В версии 2.4 константы расположены в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

Выведем значения констант:

```
System.out.println(Imgproc.LINE_4);  // 4
System.out.println(Imgproc.LINE_8);  // 8
System.out.println(Imgproc.LINE_AA); // 16
```

По умолчанию используется тип `LINE_8`, и в результате наклонная линия будет нарисована лесенкой. Чтобы при рисовании использовалось сглаживание, нужно указать тип `LINE_AA`.

Пример рисования различных линий приведен в листинге 4.2, а результат выполнения кода из листинга 4.2 показан на рис. 4.1.

Листинг 4.2. Рисование линий

```
Mat img = new Mat(300, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.line(img, new Point(50, 50), new Point(250, 50),
             CvUtils.COLOR_RED);
```

```
Imgproc.line(img, new Point(50, 100), new Point(250, 150),
             CvUtils.COLOR_BLUE, 5);
// Без сглаживания
Imgproc.line(img, new Point(50, 150), new Point(250, 200),
             CvUtils.COLOR_GREEN, 5, Imgproc.LINE_4, 0);
// Со сглаживанием
Imgproc.line(img, new Point(50, 200), new Point(250, 250),
             CvUtils.COLOR_BLACK, 5, Imgproc.LINE_AA, 0);
CvUtilsFX.showImage(img, "Рисование линий");
```

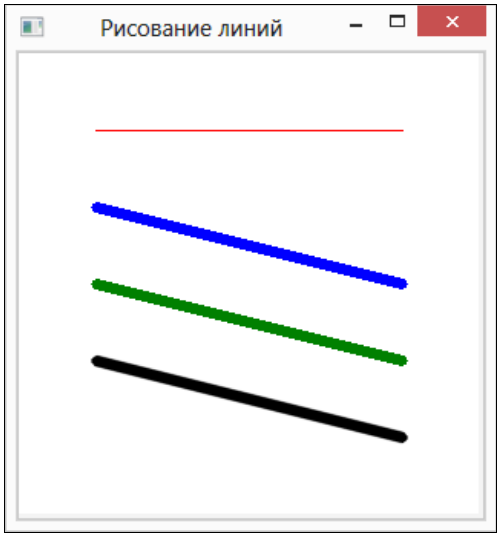


Рис. 4.1. Результат выполнения кода из листинга 4.2

4.3. Рисование стрелки

Нарисовать стрелку позволяет статический метод `arrowedLine()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void arrowedLine(Mat img, Point pt1, Point pt2,
                               Scalar color)
public static void arrowedLine(Mat img, Point pt1, Point pt2,
                               Scalar color, int thickness, int type,
                               int shift, double tipLength)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

В первом параметре указывается ссылка на изображение, во втором — координаты начальной точки, в третьем — координаты конечной точки, а в четвертом — цвет

в формате `BGR` или `Gray` (для изображений в градациях серого). По умолчанию линия рисуется толщиной в один пиксел. С помощью параметра `thickness` можно указать другую толщину. В параметре `type` указывается тип линии: константы `LINE_4`, `LINE_8` (значение по умолчанию) или `LINE_AA` (со сглаживанием), в параметре `shift` — сдвиг (значение по умолчанию: `0`), а в параметре `tipLength` — длина стрелки относительно длины линии (значение по умолчанию: `0.1`).

Пример рисования различных стрелок приведен в листинге 4.3, а результат выполнения кода из листинга 4.3 показан на рис. 4.2.

Листинг 4.3. Рисование стрелок

```
Mat img = new Mat(300, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.arrowedLine(img, new Point(50, 50), new Point(250, 50),
                    CvUtils.COLOR_RED);
Imgproc.arrowedLine(img, new Point(50, 70), new Point(250, 120),
                    CvUtils.COLOR_BLUE, 5, Imgproc.LINE_8, 0, 0.1);
// Без сглаживания
Imgproc.arrowedLine(img, new Point(50, 120), new Point(280, 180),
                    CvUtils.COLOR_GREEN, 5, Imgproc.LINE_4, 0, 0.2);
// Со сглаживанием
Imgproc.arrowedLine(img, new Point(50, 200), new Point(250, 250),
                    CvUtils.COLOR_BLACK, 5, Imgproc.LINE_AA, 0, 0.3);
CvUtilsFX.showImage(img, "Рисование стрелок");
```

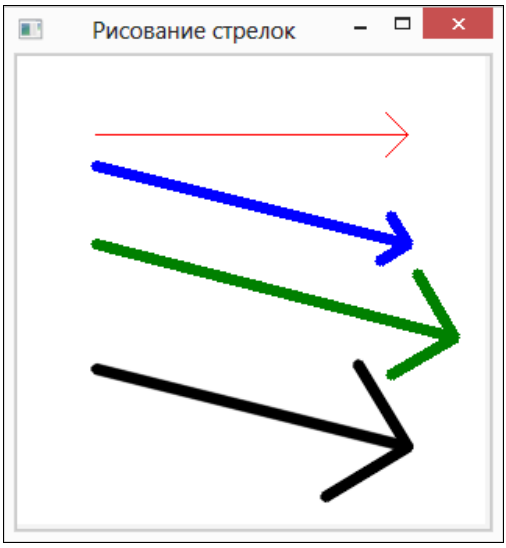


Рис. 4.2. Результат выполнения кода из листинга 4.3

4.4. Рисование прямоугольника

Нарисовать прямоугольник позволяет статический метод `rectangle()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void rectangle(Mat img, Point pt1, Point pt2, Scalar color)
public static void rectangle(Mat img, Point pt1, Point pt2, Scalar color,
                             int thickness)
public static void rectangle(Mat img, Point pt1, Point pt2, Scalar color,
                             int thickness, int lineType, int shift)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

В первом параметре указывается ссылка на изображение, во втором — координаты левого верхнего угла прямоугольника, в третьем — координаты правого нижнего угла, а в четвертом — цвет в формате `BGR` или `Gray` (для изображений в градациях серого). По умолчанию линия обводки рисуется толщиной в один пиксел. С помощью параметра `thickness` можно указать другую толщину. Если в параметре `thickness` указать константу `FILLED` из класса `Core`, то прямоугольник будет рисоваться с заливкой без обводки. Выведем значение константы:

```
System.out.println(Core.FILLED); // -1
```

В параметре `lineType` указывается тип линии: константы `LINE_4`, `LINE_8` (значение по умолчанию) или `LINE_AA` (со сглаживанием), а в параметре `shift` — сдвиг (значение по умолчанию: `0`).

Пример рисования различных прямоугольников приведен в листинге 4.4, а результат выполнения кода из листинга 4.4 показан на рис. 4.3.

Листинг 4.4. Рисование прямоугольников

```
Mat img = new Mat(300, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.rectangle(img, new Point(50, 10), new Point(250, 50),
                  CvUtils.COLOR_BLACK);
Imgproc.rectangle(img, new Point(50, 80), new Point(250, 120),
                  CvUtils.COLOR_BLUE, 15);
// Заливка без обводки
Imgproc.rectangle(img, new Point(50, 150), new Point(250, 200),
                  CvUtils.COLOR_GREEN, Core.FILLED);
// Со сглаживанием углов
Imgproc.rectangle(img, new Point(50, 230), new Point(250, 280),
                  CvUtils.COLOR_BLACK, 15, Imgproc.LINE_AA, 0);

Imgproc.rectangle(img, new Point(50, 230), new Point(250, 280),
                  CvUtils.COLOR_WHITE);
CvUtilsFX.showImage(img, "Рисование прямоугольников");
```

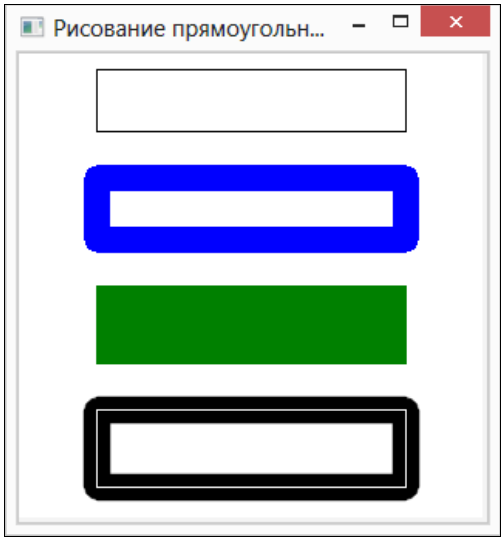


Рис. 4.3. Результат выполнения кода из листинга 4.4

4.5. Рисование круга

Нарисовать круг позволяет статический метод `circle()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void circle(Mat img, Point center, int radius,
                          Scalar color)
public static void circle(Mat img, Point center, int radius,
                          Scalar color, int thickness)
public static void circle(Mat img, Point center, int radius,
                          Scalar color, int thickness, int lineType,
                          int shift)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

В первом параметре указывается ссылка на изображение, во втором — координаты центра круга, в третьем — радиус, а в четвертом — цвет в формате `BGR` или `Gray` (для изображений в градациях серого). По умолчанию линия обводки рисуется толщиной в один пиксел. С помощью параметра `thickness` можно указать другую толщину. Если в параметре `thickness` указать константу `FILLED` из класса `Core`, то круг будет рисоваться с заливкой без обводки. В параметре `lineType` указывается тип линии: константы `LINE_4`, `LINE_8` (значение по умолчанию) или `LINE_AA` (со сглаживанием), а в параметре `shift` — сдвиг (значение по умолчанию: `0`).

Пример рисования круга приведен в листинге 4.5, а результат выполнения кода из листинга 4.5 показан на рис. 4.4.

Листинг 4.5. Рисование круга

```
Mat img = new Mat(300, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.circle(img, new Point(80, 80), 50, CvUtils.COLOR_BLACK);
Imgproc.circle(img, new Point(200, 80), 30,
               CvUtils.COLOR_BLUE, 5);
// Заливка без обводки
Imgproc.circle(img, new Point(80, 200), 50,
               CvUtils.COLOR_GREEN, Core.FILLED);
// Со сглаживанием
Imgproc.circle(img, new Point(200, 200), 50,
               CvUtils.COLOR_BLACK, 5, Imgproc.LINE_AA, 0);
CvUtilsFX.showImage(img, "Рисование круга");
```

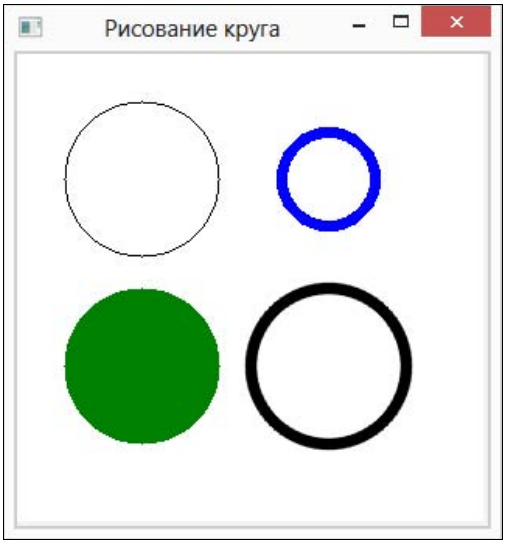


Рис. 4.4. Результат выполнения кода из листинга 4.5

4.6. Рисование эллипса, дуги или сектора

Нарисовать эллипс, дугу или сектор (начало и конец дуги соединяются через центр эллипса) позволяет статический метод `ellipse()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void ellipse(Mat img, Point center, Size axes,
       double angle, double startAngle, double endAngle, Scalar color)
public static void ellipse(Mat img, Point center, Size axes,
       double angle, double startAngle, double endAngle, Scalar color,
       int thickness)
```

```
public static void ellipse(Mat img, Point center, Size axes,
       double angle, double startAngle, double endAngle, Scalar color,
       int thickness, int lineType, int shift)
public static void ellipse(Mat img, RotatedRect box, Scalar color)
public static void ellipse(Mat img, RotatedRect box, Scalar color,
                           int thickness)
public static void ellipse(Mat img, RotatedRect box, Scalar color,
                           int thickness, int lineType)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

В первом параметре указывается ссылка на изображение, во втором — координаты центра эллипса, а в третьем — размеры (радиус по оси `X` и радиус по оси `Y`). В параметре `angle` задается угол в градусах, на который будет повернута фигура.

Начальный угол дуги задается в параметре `startAngle`, а конечный — в параметре `endAngle`. Чтобы нарисовать эллипс в параметре `startAngle`, нужно указать значение `0`, а в параметре `endAngle` — значение `360`. Нулевой угол расположен в позиции «трех часов». Увеличение угла производится по часовой стрелке.

Параметр `color` задает цвет обводки или заливки в формате `BGR` или `Gray` (для изображений в градациях серого). По умолчанию линия обводки рисуется толщиной в один пиксел. С помощью параметра `thickness` можно указать другую толщину. Если в параметре `thickness` указать константу `FILLED` из класса `Core`, то фигура будет рисоваться с заливкой без обводки. При этом концы дуги будут соединяться через центр эллипса, образуя сектор. В параметре `lineType` указывается тип линии: константы `LINE_4`, `LINE_8` (значение по умолчанию) или `LINE_AA` (со сглаживанием), а в параметре `shift` — сдвиг (значение по умолчанию: `0`).

Параметр `box` позволяет указать прямоугольную область (в виде объекта класса `RotatedRect` (см. *разд. 1.3.5*)), повернутую на некоторый угол. В этот прямоугольник будет вписан эллипс.

Пример рисования дуги и сектора приведен в листинге 4.6, а результат выполнения кода из листинга 4.6 показан на рис. 4.5.

Листинг 4.6. Рисование дуги и сектора

```
Mat img = new Mat(300, 500, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.ellipse(img, new Point(50, 60), new Size(50, 50),
                0, 0, 90, CvUtils.COLOR_BLACK);
Imgproc.ellipse(img, new Point(180, 60), new Size(50, 50),
                0, 0, 180, CvUtils.COLOR_RED, 5);
Imgproc.ellipse(img, new Point(310, 60), new Size(50, 50),
                0, 0, 270, CvUtils.COLOR_GREEN, 5, Imgproc.LINE_4, 0);
// Со сглаживанием
Imgproc.ellipse(img, new Point(440, 60), new Size(50, 50),
                0, 0, 360, CvUtils.COLOR_BLUE, 5, Imgproc.LINE_AA, 0);
```

```
Imgproc.ellipse(img, new Point(50, 200), new Size(50, 50),
                0, 0, 90, CvUtils.COLOR_BLACK, Core.FILLED);
Imgproc.ellipse(img, new Point(180, 200), new Size(50, 50),
                0, 0, 180, CvUtils.COLOR_RED, Core.FILLED);
Imgproc.ellipse(img, new Point(310, 200), new Size(50, 50),
                0, 0, 270, CvUtils.COLOR_GREEN, Core.FILLED);
// Со сглаживанием
Imgproc.ellipse(img, new Point(440, 200), new Size(50, 50),
                0, 0, 360, CvUtils.COLOR_BLUE, Core.FILLED,
                Imgproc.LINE_AA, 0);
CvUtilsFX.showImage(img, "Рисование дуги и сектора");
```

Рис. 4.5. Результат выполнения кода из листинга 4.6

Пример рисования эллипса приведен в листинге 4.7, а результат выполнения кода из листинга 4.7 показан на рис. 4.6.

Листинг 4.7. Рисование эллипса

```
Mat img = new Mat(300, 500, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.ellipse(img, new Point(60, 60), new Size(50, 30),
                0, 0, 360, CvUtils.COLOR_BLACK);
Imgproc.ellipse(img, new Point(180, 60), new Size(50, 30),
                30, 0, 360, CvUtils.COLOR_RED, 5);
Imgproc.ellipse(img, new Point(310, 60), new Size(50, 30),
                60, 0, 360, CvUtils.COLOR_GREEN, 5, Imgproc.LINE_4, 0);
// Со сглаживанием
Imgproc.ellipse(img, new Point(440, 60), new Size(30, 50),
                0, 0, 360, CvUtils.COLOR_BLUE, 5, Imgproc.LINE_AA, 0);
```

```
Imgproc.ellipse(img, new RotatedRect(new Point(60, 200),
                new Size(100, 60), 0), CvUtils.COLOR_BLACK, Core.FILLED);
Imgproc.ellipse(img, new RotatedRect(new Point(180, 200),
                new Size(100, 60), 30), CvUtils.COLOR_RED, Core.FILLED);
Imgproc.ellipse(img, new RotatedRect(new Point(310, 200),
                new Size(100, 60), 60), CvUtils.COLOR_GREEN,
                Core.FILLED);
// Со сглаживанием
Imgproc.ellipse(img, new RotatedRect(new Point(440, 200),
                new Size(60, 100), 0), CvUtils.COLOR_BLUE, Core.FILLED,
                Imgproc.LINE_AA);
CvUtilsFX.showImage(img, "Рисование эллипса");
```

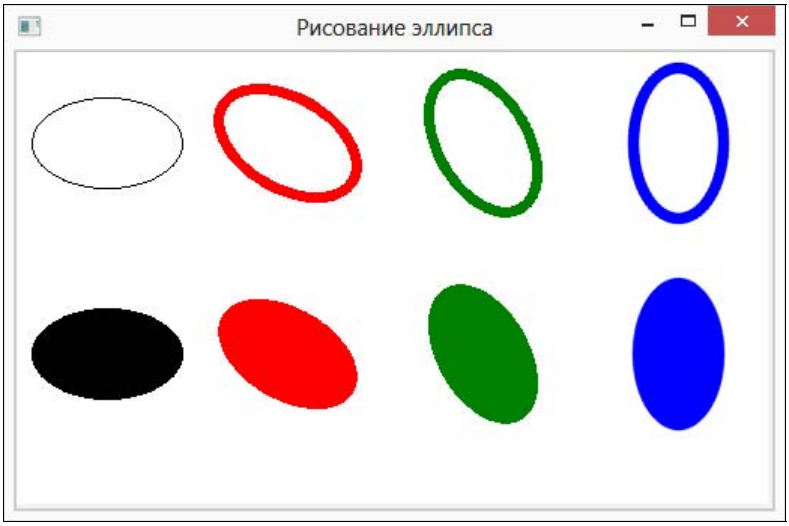


Рис. 4.6. Результат выполнения кода из листинга 4.7

4.7. Рисование ломаной линии и многоугольника

Нарисовать ломаную линию или многоугольник, используя характеристики обводки, позволяет статический метод `polylines()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void polylines(Mat img, List<MatOfPoint> pts,
                             boolean isClosed, Scalar color)
public static void polylines(Mat img, List<MatOfPoint> pts,
                  boolean isClosed, Scalar color, int thickness)
public static void polylines(Mat img, List<MatOfPoint> pts,
                  boolean isClosed, Scalar color, int thickness,
                  int lineType, int shift)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

В первом параметре указывается ссылка на изображение, а во втором — список с координатами вершин. Если в параметре `isClosed` указано значение `true`, то начальная и конечная точки будут соединены прямой линией. В параметре `color` задается цвет линии в формате `BGR` или `Gray` (для изображений в градациях серого). По умолчанию линия обводки рисуется толщиной в один пиксел. С помощью параметра `thickness` можно указать другую толщину. В параметре `lineType` указывается тип линии: константы `LINE_4`, `LINE_8` (значение по умолчанию) или `LINE_AA` (со сглаживанием), а в параметре `shift` — сдвиг (значение по умолчанию: `0`).

Пример рисования ломаной линии и треугольника приведен в листинге 4.8, а результат выполнения кода из листинга 4.8 показан на рис. 4.7.

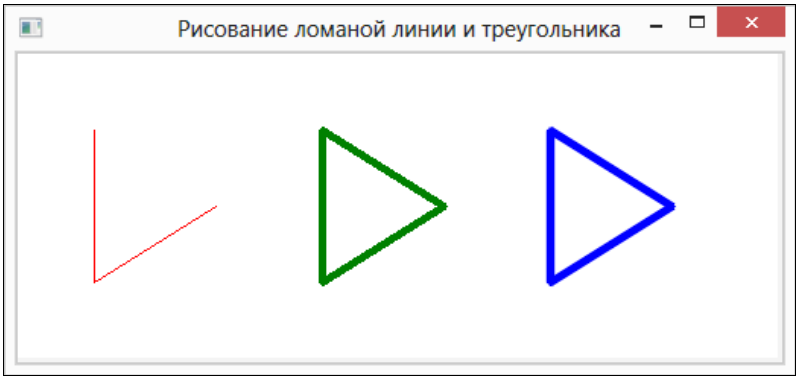


Рис. 4.7. Результат выполнения кода из листинга 4.8

Листинг 4.8. Рисование ломаной линии и треугольника

```
Mat img = new Mat(200, 500, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
MatOfPoint points = new MatOfPoint(new Point(50, 50),
                        new Point(50, 150), new Point(130, 100));
ArrayList<MatOfPoint> list = new ArrayList<MatOfPoint>();
list.add(points);
Imgproc.polylines(img, list, false, CvUtils.COLOR_RED);

MatOfPoint points2 = new MatOfPoint(new Point(200, 50),
                         new Point(200, 150), new Point(280, 100));
ArrayList<MatOfPoint> list2 = new ArrayList<MatOfPoint>();
list2.add(points2);
Imgproc.polylines(img, list2, true, CvUtils.COLOR_GREEN, 3);

MatOfPoint points3 = new MatOfPoint(new Point(350, 50),
                         new Point(350, 150), new Point(430, 100));
ArrayList<MatOfPoint> list3 = new ArrayList<MatOfPoint>();
```

```
list3.add(points3);
Imgproc.polylines(img, list3, true, CvUtils.COLOR_BLUE, 3,
                  Imgproc.LINE_AA, 0);
CvUtilsFX.showImage(img, "Рисование ломаной линии и треугольника");
```

Если нужно нарисовать многоугольник, используя характеристики заливки, то следует воспользоваться статическим методом `fillPoly()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void fillPoly(Mat img, List<MatOfPoint> pts, Scalar color)
public static void fillPoly(Mat img, List<MatOfPoint> pts, Scalar color,
                            int lineType, int shift, Point offset)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

Смысл параметров точно такой же, как и у параметров метода `polylines()`. Начальная и конечная точки будут соединены. В параметре `offset` можно указать смещение, добавляемое ко всем точкам.

Пример рисования треугольника приведен в листинге 4.9, а результат выполнения кода из листинга 4.9 показан на рис. 4.8.

Рис. 4.8. Результат выполнения кода из листинга 4.9

Листинг 4.9. Рисование треугольника с заливкой

```
Mat img = new Mat(160, 400, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
MatOfPoint points = new MatOfPoint(new Point(70, 30),
                        new Point(70, 130), new Point(180, 80));
ArrayList<MatOfPoint> list = new ArrayList<MatOfPoint>();
list.add(points);
Imgproc.fillPoly(img, list, CvUtils.COLOR_GREEN);

MatOfPoint points2 = new MatOfPoint(new Point(350, 30),
                        new Point(350, 130), new Point(480, 80));
ArrayList<MatOfPoint> list2 = new ArrayList<MatOfPoint>();
```

```
list2.add(points2);
Imgproc.fillPoly(img, list2, CvUtils.COLOR_BLUE,
              Imgproc.LINE_AA, 0, new Point(-120, 0));
CvUtilsFX.showImage(img, "Рисование треугольника с заливкой");
```

Вместо метода `fillPoly()` можно воспользоваться статическим методом `fillConvexPoly()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void fillConvexPoly(Mat img, MatOfPoint points,
                                  Scalar color)
public static void fillConvexPoly(Mat img, MatOfPoint points,
                                  Scalar color, int lineType, int shift)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

Смысл параметров точно такой же, как и у параметров метода `polylines()`. Параметр `points` задает матрицу с координатами вершин. Начальная и конечная точки будут соединены.

Пример рисования треугольника с помощью метода `fillConvexPoly()` приведен в листинге 4.10, а результат выполнения кода из листинга 4.10 показан на рис. 4.9.

Листинг 4.10. Рисование треугольника с заливкой

```
Mat img = new Mat(160, 400, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
MatOfPoint points = new MatOfPoint(new Point(70, 30),
                       new Point(20, 130), new Point(180, 80));
Imgproc.fillConvexPoly(img, points, CvUtils.COLOR_GREEN);

MatOfPoint points2 = new MatOfPoint(new Point(230, 30),
                        new Point(180, 130), new Point(360, 80));
Imgproc.fillConvexPoly(img, points2, CvUtils.COLOR_BLUE,
                      Imgproc.LINE_AA, 0);
CvUtilsFX.showImage(img, "Рисование треугольника с заливкой");
```

Рис. 4.9. Результат выполнения кода из листинга 4.10

4.8. Вывод текста на изображение

Вывести текст на изображение позволяет статический метод `putText()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void putText(Mat img, String text, Point org, int fontFace,
                           double fontScale, Scalar color)
public static void putText(Mat img, String text, Point org, int fontFace,
                           double fontScale, Scalar color, int thickness)
public static void putText(Mat img, String text, Point org, int fontFace,
                           double fontScale, Scalar color, int thickness,
                           int lineType, boolean bottomLeftOrigin)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

В первом параметре указывается ссылка на изображение, а во втором — текст, выводимый на изображение. Большинство встроенных шрифтов работают только с буквами латиницы, русские буквы, к сожалению, превращаются в знаки вопроса. Чтобы вывести текст на русском языке, следует использовать тип шрифта `FONT_HERSHEY_COMPLEX` (но вместо буквы «ё» и в этом варианте тоже выводится знак вопроса):

```
Imgproc.putText(img, "Текст", new Point(10, 100),
                Core.FONT_HERSHEY_COMPLEX, 0.7, CvUtils.COLOR_BLACK);
```

В параметре `org` задаются координаты левой крайней точки базовой линии. *Базовой* называется линия, соприкасающаяся с большинством букв снизу. Некоторые буквы имеют фрагменты, расположенные ниже базовой линии, — например, вертикальная черта у строчной буквы «р».

Тип шрифта задается в параметре `fontFace`. В качестве значения можно указать следующие константы из класса `Core`:

```
public static final int FONT_HERSHEY_PLAIN
public static final int FONT_HERSHEY_COMPLEX_SMALL
public static final int FONT_HERSHEY_SIMPLEX
public static final int FONT_HERSHEY_COMPLEX
public static final int FONT_HERSHEY_DUPLEX
public static final int FONT_HERSHEY_TRIPLEX
public static final int FONT_HERSHEY_SCRIPT_SIMPLEX
public static final int FONT_HERSHEY_SCRIPT_COMPLEX
```

Чтобы сделать шрифт наклонным, нужно дополнительно указать константу `FONT_ITALIC`:

```
public static final int FONT_ITALIC
```

Пример:

```
Imgproc.putText(img, "FONT_ITALIC", new Point(20, 20),
                Core.FONT_HERSHEY_PLAIN | Core.FONT_ITALIC, 1,
                CvUtils.COLOR_BLACK);
```

Выведем значения констант:

```
System.out.println(Core.FONT_HERSHEY_PLAIN);          // 1
System.out.println(Core.FONT_HERSHEY_COMPLEX_SMALL);  // 5
System.out.println(Core.FONT_HERSHEY_SIMPLEX);        // 0
System.out.println(Core.FONT_HERSHEY_COMPLEX);        // 3
System.out.println(Core.FONT_HERSHEY_DUPLEX);         // 2
System.out.println(Core.FONT_HERSHEY_TRIPLEX);        // 4
System.out.println(Core.FONT_HERSHEY_SCRIPT_SIMPLEX); // 6
System.out.println(Core.FONT_HERSHEY_SCRIPT_COMPLEX); // 7
System.out.println(Core.FONT_ITALIC);                 // 16
```

Параметр `fontScale` задает масштаб шрифта, а параметр `color` — цвет линии в формате `BGR` или `Gray` (для изображений в градациях серого). С помощью параметра `thickness` можно указать толщину линии. В параметре `lineType` указывается тип линии: константы `LINE_4`, `LINE_8` или `LINE_AA` (со сглаживанием). Если в параметре `bottomLeftOrigin` указать значение `true`, то текст будет зеркально отражен по вертикали.

Пример вывода текста различными шрифтами приведен в листинге 4.11, а результат выполнения кода из листинга 4.11 показан на рис. 4.10.

Листинг 4.11. Вывод текста на изображение

```
Mat img = new Mat(300, 600, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.putText(img, "OpenCV", new Point(10, 20),
                Core.FONT_HERSHEY_PLAIN, 1, CvUtils.COLOR_BLACK);
Imgproc.putText(img, "OpenCV", new Point(10, 40),
                Core.FONT_HERSHEY_PLAIN, 1, CvUtils.COLOR_BLACK, 2);
Imgproc.putText(img, "OpenCV", new Point(10, 60),
                Core.FONT_HERSHEY_PLAIN, 1, CvUtils.COLOR_BLACK, 1,
                Imgproc.LINE_AA, false);
Imgproc.putText(img, "OpenCV", new Point(10, 80),
                Core.FONT_HERSHEY_PLAIN, 1, CvUtils.COLOR_BLACK, 1,
                Imgproc.LINE_AA, true);
Imgproc.putText(img, "Текст", new Point(10, 120),
                Core.FONT_HERSHEY_COMPLEX, 0.7, CvUtils.COLOR_BLACK);

Imgproc.putText(img, "FONT_HERSHEY_PLAIN", new Point(100, 20),
                Core.FONT_HERSHEY_PLAIN, 1, CvUtils.COLOR_BLACK);
Imgproc.putText(img, "FONT_ITALIC", new Point(400, 20),
                Core.FONT_HERSHEY_PLAIN | Core.FONT_ITALIC, 1,
                CvUtils.COLOR_BLACK);
Imgproc.putText(img, "FONT_HERSHEY_COMPLEX_SMALL", new Point(100, 50),
                Core.FONT_HERSHEY_COMPLEX_SMALL, 1,
                CvUtils.COLOR_BLACK);
Imgproc.putText(img, "FONT_HERSHEY_SIMPLEX", new Point(100, 90),
                Core.FONT_HERSHEY_SIMPLEX, 1, CvUtils.COLOR_BLACK);
```

```
Imgproc.putText(img, "FONT_HERSHEY_COMPLEX", new Point(100, 120),
                Core.FONT_HERSHEY_COMPLEX, 1, CvUtils.COLOR_BLACK);
Imgproc.putText(img, "FONT_HERSHEY_DUPLEX", new Point(100, 150),
                Core.FONT_HERSHEY_DUPLEX, 1, CvUtils.COLOR_BLACK);
Imgproc.putText(img, "FONT_HERSHEY_TRIPLEX", new Point(100, 180),
                Core.FONT_HERSHEY_TRIPLEX, 1, CvUtils.COLOR_BLACK);
Imgproc.putText(img, "FONT_HERSHEY_SCRIPT_SIMPLEX", new Point(10, 220),
                Core.FONT_HERSHEY_SCRIPT_SIMPLEX, 1,
                CvUtils.COLOR_BLACK);
Imgproc.putText(img, "FONT_HERSHEY_SCRIPT_COMPLEX", new Point(10, 260),
                Core.FONT_HERSHEY_SCRIPT_COMPLEX, 1,
                CvUtils.COLOR_BLACK);

CvUtilsFX.showImage(img, "Вывод текста");
```

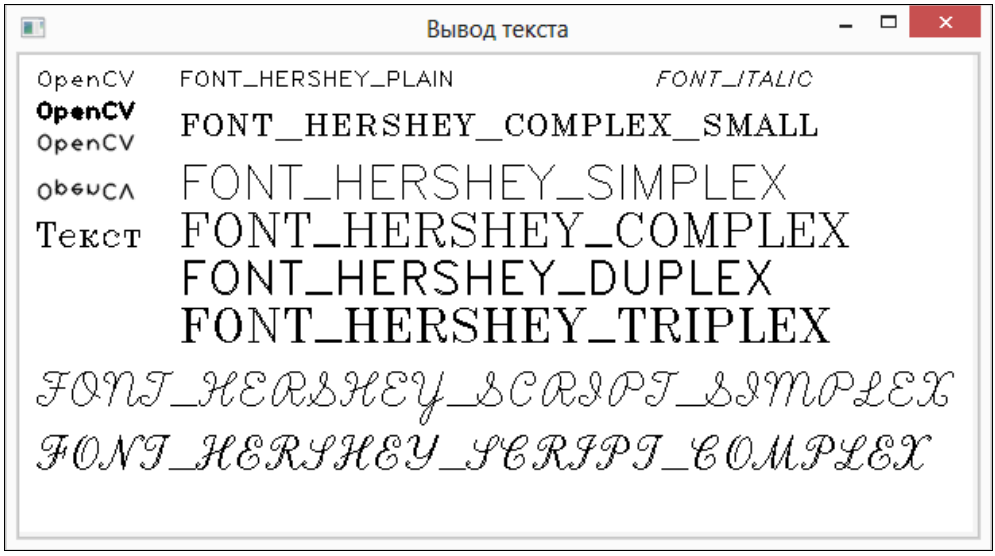


Рис. 4.10. Результат выполнения кода из листинга 4.11

С помощью статического метода `getTextSize()` из класса `Imgproc` можно рассчитать размеры прямоугольника, в который вписан текст. Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static Size getTextSize(String text, int fontFace,
                  double fontScale, int thickness, int[] baseLine)
```

В версии 2.4 метод расположен в классе `Core`. Инструкция импорта:

```
import org.opencv.core.Core;
```

Все параметры аналогичны одноименным параметрам в методе `putText()`. Метод `getTextSize()` возвращает размеры до базовой линии. Высота от базовой линии до нижней границы будет записана в первый элемент массива `baseLine`.

Пример использования метода `getTextSize()` приведен в листинге 4.12.

Листинг 4.12. Метод `getTextSize()`

```
Mat img = new Mat(100, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
int[] baseline = new int[1];
Size size = Imgproc.getTextSize("OpenCV", Core.FONT_HERSHEY_COMPLEX,
                                2, 5, baseline);
System.out.println(size);        // 255x45
System.out.println(baseline[0]); // 20

Imgproc.putText(img, "OpenCV", new Point(10, 10 + size.height),
                Core.FONT_HERSHEY_COMPLEX, 2, CvUtils.COLOR_BLACK,
                5, Imgproc.LINE_8, false);
Imgproc.rectangle(img, new Point(10, 10), new Point(10 + size.width - 1,
               10 + size.height + baseline[0] - 1), CvUtils.COLOR_RED);
CvUtilsFX.showImage(img, "getTextSize()");
```

4.9. Создание рамки

Для создания рамки внутри изображения можно воспользоваться методами, возвращающими диапазон (см. *разд. 2.1.5*), совместно с методом `setTo()` (см. *разд. 2.1.4*).

Пример добавления рамки красного цвета размером `10` пикселов приведен в листинге 4.13.

Листинг 4.13. Добавление рамки

```
Mat img = new Mat(200, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
int border = 10;
Scalar color = CvUtils.COLOR_RED;
img.rowRange(0, border).setTo(color);
img.rowRange(img.rows() - border, img.rows()).setTo(color);
img.colRange(0, border).setTo(color);
img.colRange(img.cols() - border, img.cols()).setTo(color);
CvUtilsFX.showImage(img, "Добавление рамки");
```

Для создания рамки вокруг изображения (ширина рамки добавляется к размерам изображения) можно воспользоваться методом `copyMakeBorder()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void copyMakeBorder(Mat src, Mat dst, int top, int bottom,
                                  int left, int right, int borderType)
public static void copyMakeBorder(Mat src, Mat dst, int top, int bottom,
                                  int left, int right, int borderType,
                                  Scalar value)
```

В версии 2.4 метод расположен в классе `Imgproc`. Инструкция импорта:

```
import org.opencv.imgproc.Imgproc;
```

В первом параметре указывается исходное изображение, а во втором параметре — ссылка на матрицу, в которую будет записано изображение с рамкой. Параметр `top` задает ширину верхней границы, `bottom` — ширину нижней границы, `left` — ширину левой границы, а `right` — ширину правой границы. Тип рамки указывается в параметре `borderType`. В параметре `value` можно задать цвет для рамки типа `BORDER_CONSTANT`.

В параметре `borderType` указываются следующие константы из класса `Core` (в версии 2.4 константы находятся в классе `Imgproc`):

- `BORDER_DEFAULT` — тип рамки по умолчанию, соответствует типу `BORDER_REFLECT_101`. Формат:

  ```
  public static final int BORDER_DEFAULT
  ```

- `BORDER_CONSTANT` — рамка заливается цветом, указанным в параметре `value`. Формат:

  ```
  public static final int BORDER_CONSTANT
  ```

- `BORDER_REFLECT` — зеркальное отражение. Формат:

  ```
  public static final int BORDER_REFLECT
  ```

- `BORDER_REFLECT_101` и `BORDER_REFLECT101` — зеркальное отражение. В отличие от типа `BORDER_REFLECT`, крайняя внешняя граница не копируется. Разница очень тонкая, и не сразу ее можно заметить. Однако это более естественное отражение, т. к. граница не выводится дважды. Форматы:

  ```
  public static final int BORDER_REFLECT_101
  public static final int BORDER_REFLECT101
  ```

- `BORDER_REPLICATE` — повтор крайних пикселов. Формат:

  ```
  public static final int BORDER_REPLICATE
  ```

- `BORDER_WRAP` — повтор изображения. Формат:

  ```
  public static final int BORDER_WRAP
  ```

Выведем значения этих констант:

```
System.out.println(Core.BORDER_DEFAULT);     // 4
System.out.println(Core.BORDER_CONSTANT);    // 0
System.out.println(Core.BORDER_REFLECT);     // 2
System.out.println(Core.BORDER_REFLECT_101); // 4
System.out.println(Core.BORDER_REFLECT101);  // 4
System.out.println(Core.BORDER_REPLICATE);   // 1
System.out.println(Core.BORDER_WRAP);        // 3
```

Пример добавления рамок разных типов приведен в листинге 4.14.

Листинг 4.14. Добавление внешней рамки

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto5.bmp");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
int border = 10;
Scalar color = CvUtils.COLOR_RED;
Mat img2 = new Mat();
Core.copyMakeBorder(img, img2, border, border, border, border,
                    Core.BORDER_DEFAULT);
Mat img3 = new Mat();
Core.copyMakeBorder(img, img3, border, border, border, border,
                    Core.BORDER_CONSTANT, color);
Mat img4 = new Mat();
Core.copyMakeBorder(img, img4, border, border, border, border,
                    Core.BORDER_REFLECT);
Mat img5 = new Mat();
Core.copyMakeBorder(img, img5, border, border, border, border,
                    Core.BORDER_REFLECT_101);
Mat img6 = new Mat();
Core.copyMakeBorder(img, img6, border, border, border, border,
                    Core.BORDER_REPLICATE);
Mat img7 = new Mat();
Core.copyMakeBorder(img, img7, border, border, border, border,
                    Core.BORDER_WRAP);

CvUtilsFX.showImage(img2, "BORDER_DEFAULT");
CvUtilsFX.showImage(img3, "BORDER_CONSTANT");
CvUtilsFX.showImage(img4, "BORDER_REFLECT");
CvUtilsFX.showImage(img5, "BORDER_REFLECT_101");
CvUtilsFX.showImage(img6, "BORDER_REPLICATE");
CvUtilsFX.showImage(img7, "BORDER_WRAP");
```

Если мы имеем дело с диапазоном (субматрицей), то нужно дополнительно указать флаг `BORDER_ISOLATED`. Формат:

```
public static final int BORDER_ISOLATED
```

Выведем значение константы:

```
System.out.println(Core.BORDER_ISOLATED);    // 16
```

Пример добавления рамки для субматрицы приведен в листинге 4.15.

Листинг 4.15. Добавление рамки для субматрицы

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto5.bmp");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
int border = 10;
Scalar color = CvUtils.COLOR_RED;
Mat img2 = new Mat();
Mat subMat = img.submat(100, 300, 100, 300);
Core.copyMakeBorder(subMat, img2, border, border, border, border,
                    Core.BORDER_CONSTANT | Core.BORDER_ISOLATED, color);
CvUtilsFX.showImage(img2, "BORDER_CONSTANT | BORDER_ISOLATED");
```

4.10. Вставка одного изображения в другое

Чтобы вставить одно изображение в другое, нужно выбрать диапазон (см. *разд. 2.1.5*), а затем скопировать в этот диапазон исходное изображение с помощью метода `copyTo()` из класса `Mat` (см. *разд. 2.1.6*). Размеры и тип диапазона должны соответствовать размерам и типу исходного изображения.

Пример вставки одного изображения в другое приведен в листинге 4.16, а результат выполнения кода из листинга 4.16 показан на рис. 4.11.

Листинг 4.16. Вставка одного изображения в другое

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat(70, 390, CvType.CV_8UC3, CvUtils.COLOR_BLACK);
Imgproc.putText(img2, "St. Petersburg", new Point(10, 50),
                Core.FONT_HERSHEY_PLAIN, 3, CvUtils.COLOR_WHITE,
                3, Imgproc.LINE_AA, false);
Mat roi = img.submat(new Rect(0, 0, img2.width(), img2.height()));
img2.copyTo(roi);
CvUtilsFX.showImage(img, "Результат");
img.release();
img2.release();
```

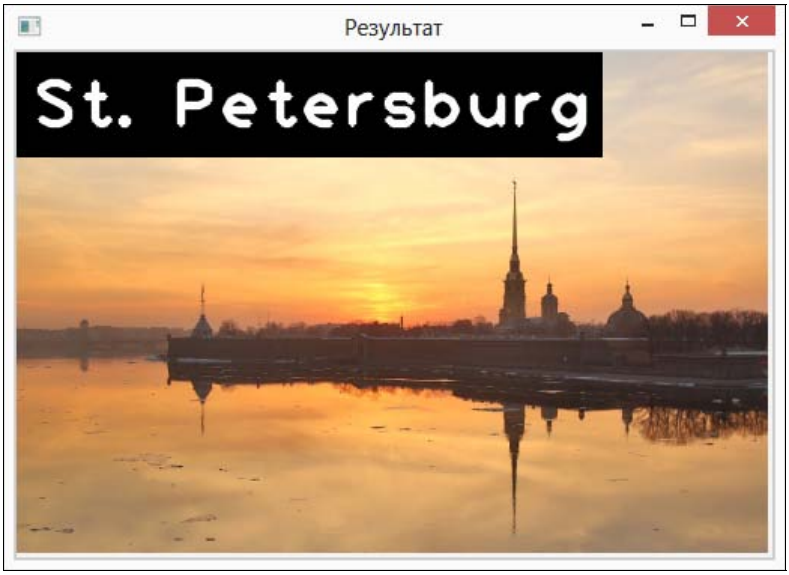


Рис. 4.11. Вставка одного изображения в другое

4.11. Наложение одного изображения на другое

Чтобы наложить одно изображение на другое, вначале нужно выбрать диапазон для вставки (см. *разд. 2.1.5*), а затем воспользоваться статическим методом `addWeighted()` из класса `Core` (см. *разд. 2.2.2*), указав процентное соотношение для каждого изображения в параметрах `alpha` и `beta`. Метод выполняет следующую операцию сложения:

```
dst(i) = src1(i) * alpha + src2(i) * beta + gamma;
```

Пример наложения одного изображения на другое приведен в листинге 4.17, а результат выполнения кода из листинга 4.17 показан на рис. 4.12.

Листинг 4.17. Наложение одного изображения на другое

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat(70, 390, CvType.CV_8UC3, CvUtils.COLOR_BLACK);
Imgproc.putText(img2, "St. Petersburg", new Point(10, 50),
                Core.FONT_HERSHEY_PLAIN, 3, CvUtils.COLOR_WHITE,
                3, Imgproc.LINE_AA, false);
Mat roi = img.submat(new Rect(0, 0, img2.width(), img2.height()));
Core.addWeighted(roi, 0.75, img2, 0.25, 0, roi);
```

```
CvUtilsFX.showImage(img, "Результат");
img.release();
img2.release();
```

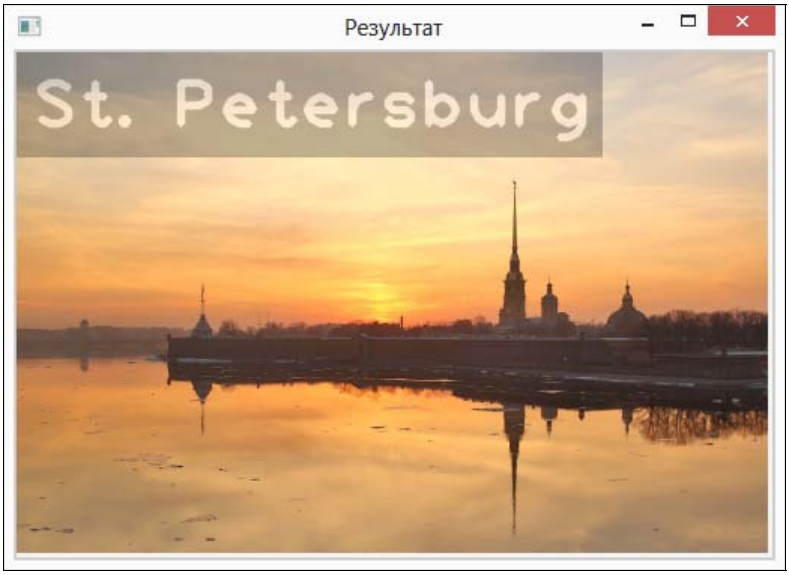


Рис. 4.12. Наложение одного изображения на другое

4.12. Рисование маркеров

Нарисовать маркеры в версии 3.3 позволяет статический метод `drawMarker()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void drawMarker(Mat img, Point position, Scalar color)
public static void drawMarker(Mat img, Point position, Scalar color,
                              int markerType, int markerSize,
                              int thickness, int line_type)
```

В первом параметре указывается ссылка на изображение, во втором — координаты места вставки маркера, в третьем — цвет в формате `BGR` или `Gray` (для изображений в градациях серого). По умолчанию линия обводки рисуется толщиной в один пиксел. С помощью параметра `thickness` можно указать другую толщину. В параметре `line_type` указывается тип линии: константы `LINE_4`, `LINE_8` (значение по умолчанию) или `LINE_AA` (со сглаживанием). Размер маркера задается в параметре `markerSize` (значение по умолчанию: `20`) а тип — в параметре `markerType` (значение по умолчанию: `MARKER_CROSS`). В параметре `markerType` можно указать следующие типы маркеров (константы из класса `Imgproc`):

```
public static final int MARKER_CROSS
public static final int MARKER_TILTED_CROSS
```

```
public static final int MARKER_STAR
public static final int MARKER_DIAMOND
public static final int MARKER_SQUARE
public static final int MARKER_TRIANGLE_UP
public static final int MARKER_TRIANGLE_DOWN
```

Пример рисования различных маркеров приведен в листинге 4.18, а результат выполнения кода из листинга 4.18 показан на рис. 4.13.

Листинг 4.18. Рисование маркеров

```
Mat img = new Mat(60, 330, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.drawMarker(img, new Point(20, 30), CvUtils.COLOR_RED);

Imgproc.drawMarker(img, new Point(60, 30), CvUtils.COLOR_BLACK,
                   Imgproc.MARKER_CROSS, 20, 1, Imgproc.LINE_8);
Imgproc.drawMarker(img, new Point(100, 30), CvUtils.COLOR_BLACK,
                   Imgproc.MARKER_DIAMOND, 20, 1, Imgproc.LINE_8);
Imgproc.drawMarker(img, new Point(140, 30), CvUtils.COLOR_BLACK,
                   Imgproc.MARKER_SQUARE, 20, 1, Imgproc.LINE_8);
Imgproc.drawMarker(img, new Point(180, 30), CvUtils.COLOR_BLACK,
                   Imgproc.MARKER_STAR, 20, 1, Imgproc.LINE_8);
Imgproc.drawMarker(img, new Point(220, 30), CvUtils.COLOR_BLACK,
                   Imgproc.MARKER_TILTED_CROSS, 20, 1, Imgproc.LINE_8);
Imgproc.drawMarker(img, new Point(260, 30), CvUtils.COLOR_BLACK,
                   Imgproc.MARKER_TRIANGLE_DOWN, 20, 1, Imgproc.LINE_8);
Imgproc.drawMarker(img, new Point(300, 30), CvUtils.COLOR_BLACK,
                   Imgproc.MARKER_TRIANGLE_UP, 20, 1, Imgproc.LINE_8);

CvUtilsFX.showImage(img, "Рисование маркеров");
```

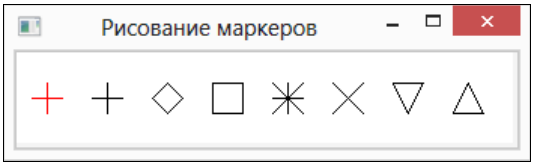
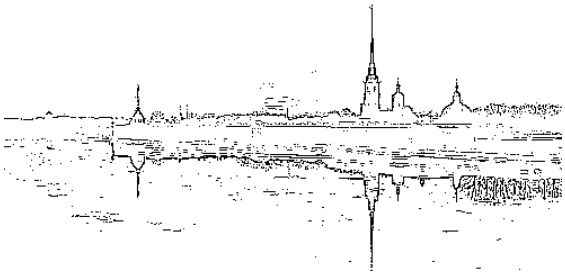


Рис. 4.13. Типы маркеров

ГЛАВА 5



Трансформация изображения

В этой главе мы рассмотрим разделение изображения на отдельные каналы и сборку изображения из отдельных каналов, а также научимся изменять размеры изображения, вращать его на нужный угол, применять аффинные преобразования и исправлять искажения перспективы.

5.1. Разделение изображения на отдельные каналы

Статический метод `split()` из класса `Core` позволяет разделить изображение на отдельные каналы. Формат метода:

```
import org.opencv.core.Core;
public static void split(Mat m, List<Mat> list)
```

В первом параметре указывается исходное изображение, а во втором параметре — ссылка на список, в который будут добавлены матрицы, состоящие из значений каждого канала. Количество элементов в списке будет равно количеству каналов в изображении (перед добавлением элементов список очищается). Первый элемент списка будет содержать значения из синего канала, второй — из зеленого, третий — из красного, а четвертый — из альфа-канала (при его наличии).

Пример использования метода `split()` приведен в листинге 5.1.

Листинг 5.1. Метод `split()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
ArrayList<Mat> list = new ArrayList<Mat>();
Core.split(img, list);
System.out.println(list.size());
```

```
int n = 0;
for (Mat m: list) {
   CvUtilsFX.showImage(m, "" + n++);
   m.release();
}
img.release();
```

Выполнить обратную операцию, т. е. собрать изображение из отдельных каналов, позволяет статический метод `merge()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void merge(List<Mat> list, Mat dst)
```

Пример:

```
Mat m = new Mat(1, 1, CvType.CV_8UC4, new Scalar(0, 128, 200, 255));
ArrayList<Mat> list = new ArrayList<Mat>();
Core.split(m, list);
System.out.println(list.get(0).dump()); // [  0]
System.out.println(list.get(1).dump()); // [128]
System.out.println(list.get(2).dump()); // [200]
System.out.println(list.get(3).dump()); // [255]
Mat m2 = new Mat();
Core.merge(list, m2);
System.out.println(m2.dump()); // [  0, 128, 200, 255]
System.out.println(m2);
// Mat [ 1*1*CV_8UC4, isCont=true, isSubmat=false,
// nativeObj=0x19daae40, dataAddr=0x89f100 ]
```

Выполнить копирование отдельных каналов позволяет статический метод `mixChannels()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void mixChannels(List<Mat> src,
                               List<Mat> dst, MatOfInt fromTo)
```

В первом параметре указывается список с матрицами, из которых будут копироваться данные, а во втором параметре — список с матрицами, куда будут вставлены данные. В параметре `fromTo` указываются пары индексов: первый элемент задает индекс канала в списке `src`, а второй элемент — индекс канала в списке `dst`. Давайте скопируем альфа-канал, изменим его значение, а затем вставим обратно (листинг 5.2).

Листинг 5.2. Копирование и вставка альфа-канала

```
Mat m = new Mat(1, 1, CvType.CV_8UC4, new Scalar(0, 128, 200, 218));
Mat alpha = new Mat(m.rows(), m.cols(), CvType.CV_8UC1);
ArrayList<Mat> listSrc = new ArrayList<Mat>();
listSrc.add(m);
ArrayList<Mat> listDst = new ArrayList<Mat>();
```

```
listDst.add(alpha);
Core.mixChannels(listSrc, listDst, new MatOfInt(3, 0));
System.out.println(alpha.dump());       // [218]
System.out.println(alpha);
// Mat [ 1*1*CV_8UC1, isCont=true, isSubmat=false,
// nativeObj=0x1a616100, dataAddr=0x11cb580 ]
alpha.put(0, 0, 111);
listSrc.clear();
listSrc.add(alpha);
listDst.clear();
listDst.add(m);
Core.mixChannels(listSrc, listDst, new MatOfInt(0, 3));
System.out.println(m.dump());       // [  0, 128, 200, 111]
```

Скопировать значения какого-либо канала в отдельную матрицу позволяет статический метод `extractChannel()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void extractChannel(Mat src, Mat dst, int coi)
```

В первом параметре указывается исходная матрица, во втором — матрица, в которую будут скопированы значения, а в третьем — индекс канала, из которого будут скопированы значения.

Вставить значения в какой-либо канал позволяет статический метод `insertChannel()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void insertChannel(Mat src, Mat dst, int coi)
```

В первом параметре указывается матрица, из которой будут копироваться значения, во втором — матрица, в которую будут вставлены значения, а в третьем — индекс канала в матрице `dst`, в который будут вставлены значения.

Пример:

```
Mat m = new Mat(1, 3, CvType.CV_8UC3);
m.put(0, 0,
      1, 2, 3,
      4, 5, 6,
      7, 8, 9
);
Mat ch = new Mat();
// Извлечение значений каналов
Core.extractChannel(m, ch, 0);
System.out.println(m.dump());
// [  1,   2,   3,   4,   5,   6,   7,   8,   9]
System.out.println(ch.dump()); // [  1,   4,   7]
Core.extractChannel(m, ch, 1);
System.out.println(ch.dump()); // [  2,   5,   8]
```

```
Core.extractChannel(m, ch, 2);
System.out.println(ch.dump()); // [  3,   6,   9]
// Вставка значений в канал с индексом 1
Core.insertChannel(ch, m, 1);
System.out.println(m.dump());
// [  1,   3,   3,   4,   6,   6,   7,   9,   9]
```

5.2. Создание зеркального отражения

Создать зеркальное отражение изображения по горизонтали, вертикали или и по горизонтали, и по вертикали позволяет статический метод `flip()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void flip(Mat src, Mat dst, int flipCode)
```

В первом параметре указывается исходное изображение, а во втором параметре — ссылка на матрицу, в которую будет записано измененное изображение. Параметр `flipCode` задает направление отражения:

- `положительное число` — отражение по горизонтали;
- `0` — отражение по вертикали;
- `отрицательное число` — отражение и по горизонтали, и по вертикали.

Пример использования метода `flip()` приведен в листинге 5.3.

Листинг 5.3. Зеркальное отражение

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Core.flip(img, img2, 0);
Mat img3 = new Mat();
Core.flip(img, img3, 1);
Mat img4 = new Mat();
Core.flip(img, img4, -1);
CvUtilsFX.showImage(img2, "flipCode = 0");
CvUtilsFX.showImage(img3, "flipCode = 1");
CvUtilsFX.showImage(img4, "flipCode = -1");
img.release();
img2.release();
img3.release();
img4.release();
```

5.3. Объединение нескольких изображений

Объединить несколько изображений в одно позволяют статические методы `hconcat()` и `vconcat()` из класса `Core`. Форматы методов:

```
import org.opencv.core.Core;
public static void hconcat(List<Mat> src, Mat dst)
public static void vconcat(List<Mat> src, Mat dst)
```

Метод `hconcat()` объединяет изображения из списка `src` по горизонтали, а метод `vconcat()` — по вертикали. Результат записывается в матрицу `dst`.

Пример объединения изображения с зеркальными образами по горизонтали и вертикали приведен в листинге 5.4.

Листинг 5.4. Объединение изображений

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Core.flip(img, img2, 0);
Mat img3 = new Mat();
Core.flip(img, img3, 1);

ArrayList<Mat> listH = new ArrayList<Mat>();
listH.add(img);
listH.add(img3);

ArrayList<Mat> listV = new ArrayList<Mat>();
listV.add(img);
listV.add(img2);

Mat imgH = new Mat();
Core.hconcat(listH, imgH);
Mat imgV = new Mat();
Core.vconcat(listV, imgV);

CvUtilsFX.showImage(imgH, "hconcat");
CvUtilsFX.showImage(imgV, "vconcat");
img.release();
img2.release();
img3.release();
imgH.release();
imgV.release();
```

5.4. Повтор изображения по горизонтали и вертикали

Повторить изображение несколько раз по горизонтали и вертикали (выполнить заливку текстурой) позволяет статический метод `repeat()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void repeat(Mat src, int ny, int nx, Mat dst)
```

В первом параметре указывается исходное изображение, во втором — количество повторов по вертикали, в третьем — количество повторов по горизонтали, а в четвертом — матрица, в которую будет записан результат. Если изображение в какой-либо плоскости повторять не нужно, то в качестве значения указывается число `1`.

Пример использования метода `repeat()` показан в листинге 5.5.

Листинг 5.5. Повтор изображения по горизонтали и вертикали

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Core.repeat(img, 2, 1, img2);
CvUtilsFX.showImage(img2, "repeat(img, 2, 1, img2)");
Mat img3 = new Mat();
Core.repeat(img, 2, 2, img3);
CvUtilsFX.showImage(img3, "repeat(img, 2, 2, img3)");
img.release();
img2.release();
img3.release();
```

5.5. Изменение размеров изображения

Изменить размеры изображения позволяет статический метод `resize()` из класса `Imgproc`. Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static void resize(Mat src, Mat dst, Size dsize)
public static void resize(Mat src, Mat dst, Size dsize,
                          double fx, double fy, int interpolation)
```

В первом параметре указывается исходное изображение, а во втором параметре — ссылка на матрицу, в которую будет записано измененное изображение. Параметр `dsize` задает новые размеры изображения. Если указанные размеры не пропорциональны, то изображение будет искажено.

Пример:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto2.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.resize(img, img2, new Size(1280, 720));
CvUtilsFX.showImage(img2, "resize");
System.out.println(img2);
// Mat [ 720*1280*CV_8UC3, isCont=true, isSubmat=false,
// nativeObj=0x1972e7f0, dataAddr=0x1c0bd050 ]
img.release();
img2.release();
```

В параметре `fx` можно указать коэффициент масштабирования по оси `X`, а в параметре `fy` — по оси `Y`. Коэффициенты используются только в том случае, если размеры, указанные в третьем параметре, равны нулю. Если коэффициенты равны нулю, то обязательно должны быть указаны новые размеры в параметре `dsize`.

Параметр `interpolation` задает метод интерполяции, с помощью которого будут изменяться размеры изображения. Можно указать следующие константы из класса `Imgproc`:

- `INTER_NEAREST` — интерполяция методом ближайшего соседа. При уменьшении изображения наклонные линии будут отображаться «лесенкой», а при увеличении — пикселы просто повторяются. Это самый быстрый алгоритм изменения размеров. Формат:

  ```
  public static final int INTER_NEAREST
  ```

- `INTER_AREA` — дает более качественный результат из всех методов при уменьшении изображения. При увеличении результат похож на результат метода `INTER_NEAREST`. По времени выполняется дольше, чем описанные далее билинейная и бикубическая интерполяции. Формат:

  ```
  public static final int INTER_AREA
  ```

- `INTER_LINEAR` — билинейная интерполяция (используется по умолчанию). При уменьшении изображения немного уступает в качестве методу `INTER_AREA`, но выполняется почти в два раза быстрее. При увеличении изображения по качеству уступает методу `INTER_CUBIC`, но выполняется почти в два раза быстрее. Пожалуй, самый оптимальный алгоритм изменения размеров по соотношению скорости выполнения и качества. Формат:

  ```
  public static final int INTER_LINEAR
  ```

- `INTER_CUBIC` — бикубическая интерполяция (окрестность 4 на 4 пиксела). Самый лучший метод для увеличения изображения. Формат:

  ```
  public static final int INTER_CUBIC
  ```

- INTER_LANCZOS4 — фильтр Ланцоша (окрестность 8 на 8 пикселов). Самый долгий метод по времени выполнения. Формат:

```
public static final int INTER_LANCZOS4
```

Выведем значения констант:

```
System.out.println(Imgproc.INTER_NEAREST);  // 0
System.out.println(Imgproc.INTER_AREA);     // 3
System.out.println(Imgproc.INTER_LINEAR);   // 1
System.out.println(Imgproc.INTER_CUBIC);    // 2
System.out.println(Imgproc.INTER_LANCZOS4); // 4
```

Запустив код из листинга 5.6, вы сможете наглядно увидеть разницу в качестве при изменении размеров разными методами интерполяции.

Листинг 5.6. Изменение размеров изображения

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto2.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
double fx = 1280.0 / img.width();
double fy = 1280.0 / img.width();
Mat img2 = new Mat();
Imgproc.resize(img, img2, new Size(), fx, fy,
               Imgproc.INTER_NEAREST);
Mat img3 = new Mat();
Imgproc.resize(img, img3, new Size(), fx, fy,
               Imgproc.INTER_AREA);
Mat img4 = new Mat();
Imgproc.resize(img, img4, new Size(), fx, fy,
               Imgproc.INTER_LINEAR);
Mat img5 = new Mat();
Imgproc.resize(img, img5, new Size(), fx, fy,
               Imgproc.INTER_CUBIC);
Mat img6 = new Mat();
Imgproc.resize(img, img6, new Size(), fx, fy,
               Imgproc.INTER_LANCZOS4);
CvUtilsFX.showImage(img2, "INTER_NEAREST");
CvUtilsFX.showImage(img3, "INTER_AREA");
CvUtilsFX.showImage(img4, "INTER_LINEAR");
CvUtilsFX.showImage(img5, "INTER_CUBIC");
CvUtilsFX.showImage(img6, "INTER_LANCZOS4");
img.release();
img2.release();
img3.release();
```

```
img4.release();
img5.release();
img6.release();
```

5.6. Аффинные преобразования

Статический метод `warpAffine()` из класса `Imgproc` позволяет выполнить различные трансформации изображения (смещение, масштабирование, вращение и сдвиг). Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void warpAffine(Mat src, Mat dst, Mat M, Size dsize)
public static void warpAffine(Mat src, Mat dst, Mat M, Size dsize,
                  int flags)
public static void warpAffine(Mat src, Mat dst, Mat M, Size dsize,
                  int flags, int borderMode, Scalar borderValue)
```

В первом параметре указывается исходное изображение, а во втором — матрица, в которую будет записан результат операции. Параметр `M` задает матрицу трансформации размером `2x3`. Матрица имеет следующий вид:

```
mxx  mxy  tx
myx  myy  ty
```

Вычисление координат с учетом трансформации производится так:

```
x = mxx * x + mxy * y + tx
y = myx * x + myy * y + ty
```

5.6.1. Смещение, масштабирование и сдвиг

Пример матрицы трансформации для смещения на `30 px` по горизонтали и `20 px` по вертикали:

```
Mat M = new Mat(2, 3, CvType.CV_32FC1);
M.put(0, 0,
      1, 0, 30,
      0, 1, 20
);
```

Пример матрицы трансформации для масштабирования в два раза и смещения на `10 px` по горизонтали и вертикали:

```
Mat M2 = new Mat(2, 3, CvType.CV_32FC1);
M2.put(0, 0,
      2, 0, 10,
      0, 2, 10
);
```

Пример матрицы трансформации для сдвига по горизонтали и смещения на `10 px` по горизонтали и вертикали:

```
Mat M3 = new Mat(2, 3, CvType.CV_32FC1);
M3.put(0, 0,
   1, 0.1, 10,
   0, 1,   10
);
```

Создать матрицу трансформации позволяет статический метод `getAffineTransform()` из класса `Imgproc`. Формат метода:

```
public static Mat getAffineTransform(MatOfPoint2f src, MatOfPoint2f dst)
```

В первом параметре указывается список с координатами исходных трех точек, а во втором — положение этих точек после трансформации. Метод вернет матрицу трансформации.

Пример:

```
MatOfPoint2f src = new MatOfPoint2f(
      new Point(50, 50), new Point(200, 50), new Point(50, 200));
MatOfPoint2f dst = new MatOfPoint2f(
      new Point(10, 100), new Point(200, 50), new Point(100, 250));
Mat M4 = Imgproc.getAffineTransform(src, dst);
System.out.println(M4.dump());
/*
[1.266666666666667, 0.6, -83.33333333333334;
-0.3333333333333334, 1, 66.66666666666669]*/
```

В параметре `dsize` в методе `warpAffine()` указываются размеры нового изображения. Параметр `flags` задает метод интерполяции (методы интерполяции см. в *разд. 5.5*). Совместно с методом интерполяции можно указать флаг `WARP_INVERSE_MAP` (константа из класса `Imgproc`), который означает обратное преобразование. Формат:

```
public static final int WARP_INVERSE_MAP
```

Создать матрицу обратного преобразования позволяет статический метод `invertAffineTransform()` из класса `Imgproc`. Формат метода:

```
public static void invertAffineTransform(Mat M, Mat iM)
```

Пример:

```
Mat M = new Mat(2, 3, CvType.CV_32FC1);
M.put(0, 0,
   1, 0.1, 10,
   0, 1,   10
);
Mat M2 = new Mat(2, 3, CvType.CV_32FC1);
Imgproc.invertAffineTransform(M, M2);
System.out.println(M2.dump());
/*
[1, -0.1, -9;
-0, 1, -10]*/
```

Параметр `borderMode` в методе `warpAffine()` задает тип рамки (см. *разд. 4.9*), а параметр `borderValue` — цвет при использовании типа `BORDER_CONSTANT`. Если в качестве типа рамки указать константу `BORDER_TRANSPARENT`, то рамка будет прозрачной (при условии наличия в изображении альфа-канала). Пример использования метода `warpAffine()` приведен в листинге 5.7.

Листинг 5.7. Метод `warpAffine()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
// Трансформация смещения
Mat M = new Mat(2, 3, CvType.CV_32FC1);
M.put(0, 0,
      1, 0, 30,
      0, 1, 20
);
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2BGRA);
Mat img2 = new Mat();
Imgproc.warpAffine(img, img2, M,
      new Size(img.width() + 30*2, img.height() + 20*2),
      Imgproc.INTER_LINEAR, Core.BORDER_TRANSPARENT,
      new Scalar(0, 0, 0, 255));
CvUtilsFX.showImage(img2, "Смещение");
// Трансформация масштабирования
Mat M2 = new Mat(2, 3, CvType.CV_32FC1);
M2.put(0, 0,
      2, 0, 10,
      0, 2, 10
);
Mat img3 = new Mat();
Imgproc.warpAffine(img, img3, M2,
      new Size(img.width()*2 + 20, img.height()*2 + 20),
      Imgproc.INTER_CUBIC, Core.BORDER_CONSTANT,
      new Scalar(0, 0, 255, 255));
CvUtilsFX.showImage(img3, "Масштабирование");
// Трансформация сдвига
Mat M3 = new Mat(2, 3, CvType.CV_32FC1);
M3.put(0, 0,
   1, 0.1, 10,
   0, 1, 10
);
Mat img4 = new Mat();
Imgproc.warpAffine(img, img4, M3,
   new Size(img.width() * 1.5, img.height() * 1.5),
```

```
   Imgproc.INTER_LINEAR, Core.BORDER_CONSTANT,
   new Scalar(255, 0, 0, 255));
CvUtilsFX.showImage(img4, "Сдвиг");
// Метод getAffineTransform()
MatOfPoint2f src = new MatOfPoint2f(
      new Point(50, 50), new Point(200, 50), new Point(50, 200));
MatOfPoint2f dst = new MatOfPoint2f(
      new Point(10, 100), new Point(200, 50), new Point(100, 250));
Mat M4 = Imgproc.getAffineTransform(src, dst);
Mat img5 = new Mat();
Imgproc.warpAffine(img, img5, M4,
   new Size(img.width() * 1.5, img.height() * 1.5),
   Imgproc.INTER_LINEAR, Core.BORDER_CONSTANT,
   new Scalar(0, 128, 0, 255));
CvUtilsFX.showImage(img5, "getAffineTransform");
img.release();  M.release();
img2.release(); M2.release();
img3.release(); M3.release();
img4.release(); M4.release();
img5.release();
```

5.6.2. Вращение

Создать матрицу трансформации для вращения позволяет статический метод `getRotationMatrix2D()` из класса `Imgproc`. Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static Mat getRotationMatrix2D(Point center, double angle,
                                      double scale)
```

В первом параметре указываются координаты точки, вокруг которой производится вращение, во втором — угол в градусах (положительное значение означает поворот против часовой стрелки), а в третьем — коэффициент масштабирования. Матрица вращения выглядит следующим образом:

```
 a    b     (1 - a) * center.x - b * center.y
-b    a     b * center.x + (1 - a) * center.y
```

где `a` и `b` вычисляются так:

```
a = scale * cos(angle)
b = scale * sin(angle)
```

Пример вращения на 45 градусов относительно центра приведен в листинге 5.8. Как можно видеть на рис. 5.1, в результате вращения часть изображения выходит за границы и обрезается. Чтобы этого не происходило, нужно рассчитать размеры и положение прямоугольной области после вращения, а затем внести корректировку в матрицу вращения. Выполнить эти расчеты позволяет класс `RotatedRect` (см. *разд. 1.3.5*).

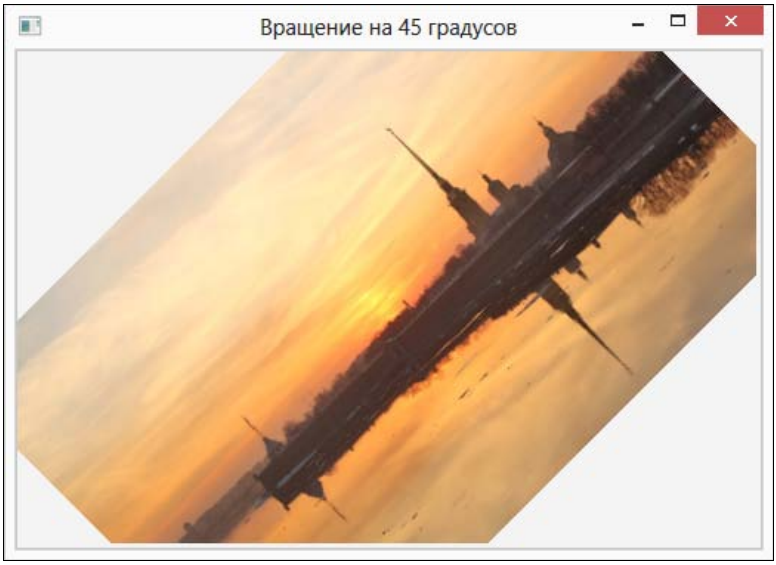


Рис. 5.1. Результат вращения на 45 градусов (листинг 5.8)

Пример вращения изображения без обрезки углов приведен в листинге 5.9, а результат выполнения кода из листинга 5.9 показан на рис. 5.2.

Листинг 5.8. Вращение изображения относительно центра

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2BGRA);
// Трансформация вращения
Mat M = Imgproc.getRotationMatrix2D(
        new Point(img.width() / 2, img.height() / 2), 45, 1);
Mat img2 = new Mat();
Imgproc.warpAffine(img, img2, M,
      new Size(img.width(), img.height()),
      Imgproc.INTER_LINEAR, Core.BORDER_TRANSPARENT,
      new Scalar(0, 0, 0, 255));
CvUtilsFX.showImage(img2, "Вращение на 45 градусов");
Mat M2 = Imgproc.getRotationMatrix2D(
         new Point(img.width() / 2, img.height() / 2), -45, 0.5);
Mat img3 = new Mat();
Imgproc.warpAffine(img, img3, M2,
      new Size(img.width(), img.height()),
      Imgproc.INTER_LINEAR, Core.BORDER_CONSTANT,
      new Scalar(0, 0, 255, 255));
```

```
CvUtilsFX.showImage(img3, "Вращение на -45 градусов, масштаб 0.5");
img.release();  M.release();
img2.release(); M2.release();
img3.release();
```

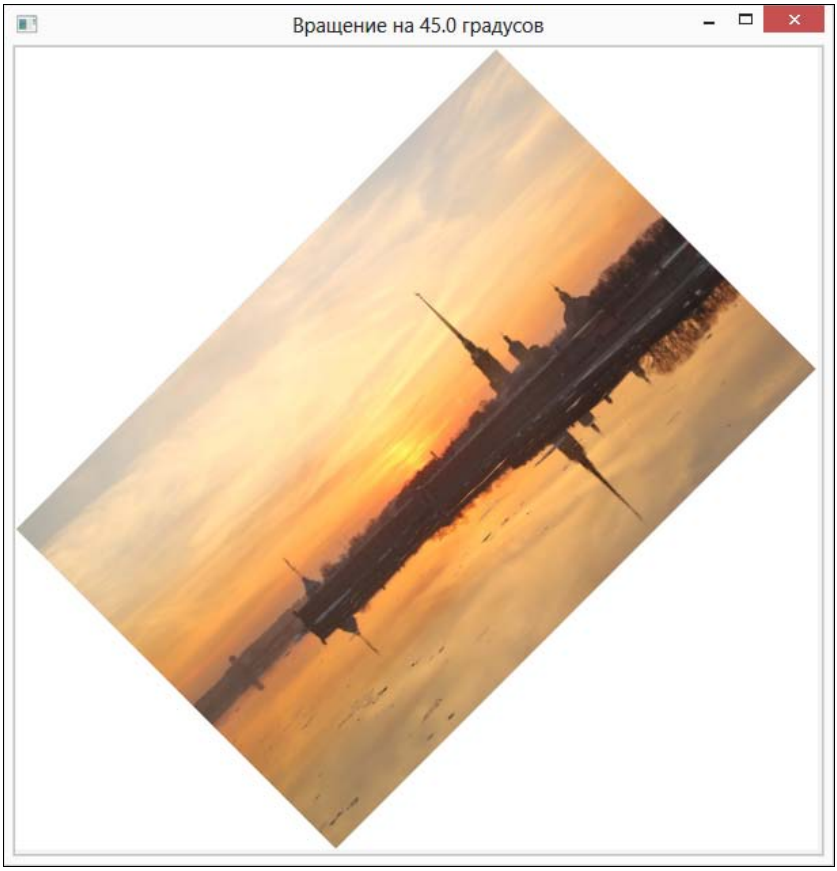


Рис. 5.2. Вращение на 45 градусов без обрезки углов (листинг 5.9)

Листинг 5.9. Вращение изображения относительно центра без обрезки углов

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2BGRA);
// Трансформация вращения
double angle = 45;
Mat M = Imgproc.getRotationMatrix2D(
         new Point(img.width() / 2, img.height() / 2), angle, 1);
```

```
// Расчет размеров и положения
Rect rect = new RotatedRect(
         new Point(img.width() / 2, img.height() / 2),
         new Size(img.width(), img.height()), angle).boundingRect();
// Корректировка матрицы трансформации
double[] arrX = M.get(0, 2);
double[] arrY = M.get(1, 2);
arrX[0] -= rect.x;
arrY[0] -= rect.y;
M.put(0, 2, arrX);
M.put(1, 2, arrY);
// Трансформация
Mat img2 = new Mat();
Imgproc.warpAffine(img, img2, M, rect.size(),
      Imgproc.INTER_LINEAR, Core.BORDER_CONSTANT,
      new Scalar(255, 255, 255, 255));
CvUtilsFX.showImage(img2, "Вращение на " + angle + " градусов");
img.release();  M.release();
img2.release();
```

Повернуть изображение на `90`, `180` или `270` градусов позволяет также статический метод `rotate()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void rotate(Mat src, Mat dst, int rotateCode) // Нет в 2.4
```

В первом параметре указывается исходное изображение, а во втором — матрица, в которую будет записан результат операции. В параметре `rotateCode` можно указать следующие константы из класса `Core`:

```
public static final int ROTATE_90_CLOCKWISE
public static final int ROTATE_180
public static final int ROTATE_90_COUNTERCLOCKWISE
```

Выведем значения констант:

```
System.out.println(Core.ROTATE_90_CLOCKWISE);        // 0
System.out.println(Core.ROTATE_180);                 // 1
System.out.println(Core.ROTATE_90_COUNTERCLOCKWISE); // 2
```

Пример:

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Core.rotate(img, img2, Core.ROTATE_180);
CvUtilsFX.showImage(img2, "ROTATE_180");
```

```
Mat img3 = new Mat();
Core.rotate(img, img3, Core.ROTATE_90_CLOCKWISE);
CvUtilsFX.showImage(img3, "ROTATE_90_CLOCKWISE");
Mat img4 = new Mat();
Core.rotate(img, img4, Core.ROTATE_90_COUNTERCLOCKWISE);
CvUtilsFX.showImage(img4, "ROTATE_90_COUNTERCLOCKWISE");
img.release();   img2.release();
img3.release();  img4.release();
```

5.7. Трансформация перспективы

Выполнить трансформацию перспективы позволяет статический метод `warpPerspective()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void warpPerspective(Mat src, Mat dst, Mat M, Size dsize)
public static void warpPerspective(Mat src, Mat dst, Mat M, Size dsize,
                       int flags)
public static void warpPerspective(Mat src, Mat dst, Mat M, Size dsize,
                       int flags, int borderMode, Scalar borderValue)
```

В первом параметре указывается исходное изображение, а во втором — матрица, в которую будет записан результат операции. Параметр `M` задает матрицу трансформации размером `3x3`. Вычисление выглядит следующим образом:

```
x = (M_11 * x + M_12 * y + M_13) / (M_31 * x + M_32 * y + M_33)
y = (M_21 * x + M_22 * y + M_23) / (M_31 * x + M_32 * y + M_33)
```

В параметре `dsize` указываются размеры нового изображения. Параметр `flags` задает метод интерполяции (`INTER_LINEAR` или `INTER_NEAREST`). Совместно с методом интерполяции можно указать флаг `WARP_INVERSE_MAP` (константа из класса `Imgproc`), который означает обратное преобразование. Параметр `borderMode` задает тип рамки (`BORDER_CONSTANT` или `BORDER_REPLICATE`), а параметр `borderValue` — цвет при использовании типа `BORDER_CONSTANT`.

Создать матрицу трансформации позволяет статический метод `getPerspectiveTransform()` из класса `Imgproc`. Формат метода:

```
public static Mat getPerspectiveTransform(Mat src, Mat dst)
```

Параметр `src` задает матрицу с координатами четырех углов в исходном изображении, а параметр `dst` — матрицу с координатами этих же четырех углов в итоговом изображении. Метод возвращает матрицу трансформации.

> ***Примечание***
>
> Создать матрицу трансформации на основе множества точек позволяет статический метод `findHomography()` из класса `Calib3d` (см. *разд. 10.4*).

Пример использования метода `warpPerspective()` приведен в листинге 5.10, а результат выполнения кода из листинга 5.10 показан на рис. 5.3.

Листинг 5.10. Метод `warpPerspective()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2BGRA);
Mat src = new Mat(4, 1, CvType.CV_32FC2);
src.put(0, 0,
      0, 0,                              // Левый верхний угол
      img.width() - 1, 0,                // Правый верхний угол
      0, img.height() - 1,               // Левый нижний угол
      img.width() - 1, img.height() - 1 // Правый нижний угол
);
Mat dst = new Mat(4, 1, CvType.CV_32FC2);
dst.put(0, 0,
      100, 100,                          // Левый верхний угол
      img.width() - 101, 100,            // Правый верхний угол
      0, img.height() - 1,               // Левый нижний угол
      img.width() - 1, img.height() - 1 // Правый нижний угол
);
Mat M = Imgproc.getPerspectiveTransform(src, dst);
Mat img2 = new Mat();
Imgproc.warpPerspective(img, img2, M, img.size(),
      Imgproc.INTER_LINEAR, Core.BORDER_CONSTANT,
      new Scalar(255, 255, 255, 255));
CvUtilsFX.showImage(img2, "Трансформация перспективы");
img.release();  M.release();
img2.release(); src.release(); dst.release();
```

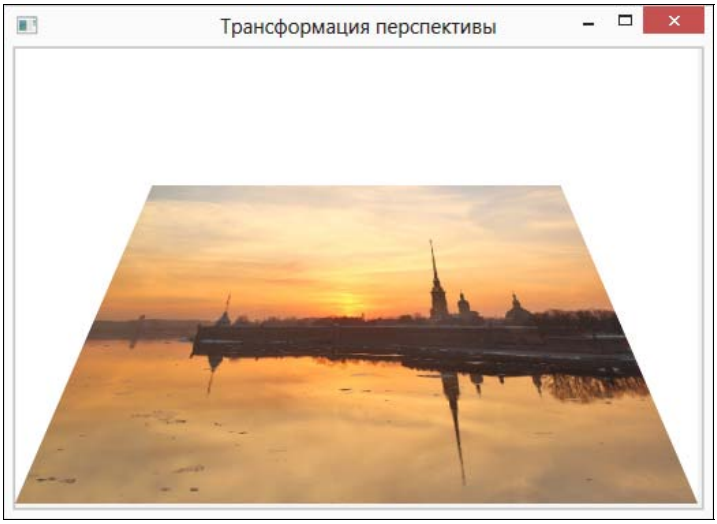
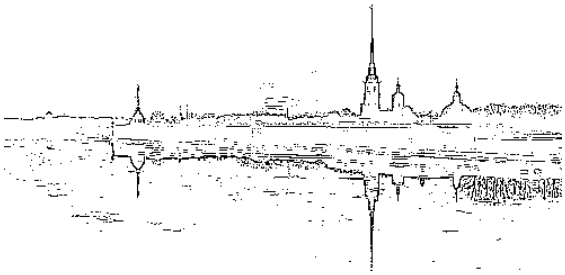


Рис. 5.3. Трансформация перспективы



ГЛАВА 6

Изменение значений компонентов цвета

Прежде чем выполнять поиск объектов на изображении, часто бывает нужно произвести предварительную обработку изображения — например, поднять контраст, повысить или понизить яркость и т. д. Чтобы понять, нужна ли изображению обработка, следует вычислить его гистограмму. *Гистограмма* — это статистическая информация о количестве пикселов, имеющих одинаковое значение. При правильно выбранной экспозиции мы получим плавное возрастание значений до середины гистограммы, а затем плавное их уменьшение. Если пик расположен ближе к ее левой или правой границе, то изображение либо нуждается в обработке, либо просто снимок сделан в специфических условиях: ночью или зимой (при наличии снега).

6.1. Преобразование цветного изображения в черно-белое

В *разд. 3.4.1* мы рассмотрели способ преобразования цветного изображения в изображение в оттенках серого с помощью статического метода `cvtColor()` из класса `Imgproc`:

```
Mat m = new Mat(1, 1, CvType.CV_8UC3, new Scalar(0, 128, 255));
System.out.println(m.dump()); // [  0, 128, 255]
Mat m2 = new Mat();
Imgproc.cvtColor(m, m2, Imgproc.COLOR_BGR2GRAY);
System.out.println(m2.dump()); // [151]
```

> ***Примечание***
>
> Преобразовать цветное изображение в изображение в оттенках серого можно также с помощью обнуления значения канала `S` (насыщенность) в цветовой модели `HSV` (см. *разд. 6.2*).

Обратите внимание: изображение в оттенках серого не является черно-белым изображением. В первом случае используется 256 градаций серого цвета, а во втором — только два цвета: черный и белый.

Преобразовать изображение в оттенках серого в черно-белое позволяет статический метод `threshold()` из класса `Imgproc`. Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static double threshold(Mat src, Mat dst, double thresh,
                               double maxval, int type)
```

В первом параметре указывается исходное изображение, а во втором параметре — ссылка на матрицу, в которую будет записано измененное изображение. Элементы исходного изображения должны состоять из одного канала и иметь глубину цвета 8 битов или 32 бита (вещественное число). Параметр `thresh` задает пороговое значение, а параметр `maxval` — максимальное значение (используется только в алгоритмах `THRESH_BINARY` и `THRESH_BINARY_INV`). В параметре `type` можно указать следующие алгоритмы (константы из класса `Imgproc`):

- `THRESH_BINARY` — если `src(x, y)` больше `thresh`, то `dst(x, y)` будет иметь значение `maxval`, в противном случае `0`. Формат:

  ```
  public static final int THRESH_BINARY
  ```

  Пример:

  ```
  Mat m = new Mat(1, 3, CvType.CV_8UC1);
  m.put(0, 0, 50.0, 127.0, 220.0);
  System.out.println(m.dump()); // [ 50, 127, 220]
  Mat m2 = new Mat();
  Imgproc.threshold(m, m2, 100, 255, Imgproc.THRESH_BINARY);
  System.out.println(m2.dump()); // [  0, 255, 255]
  ```

- `THRESH_BINARY_INV` — если `src(x, y)` больше `thresh`, то `dst(x, y)` будет иметь значение `0`, в противном случае `maxval`. Формат:

  ```
  public static final int THRESH_BINARY_INV
  ```

  Пример:

  ```
  Mat m = new Mat(1, 3, CvType.CV_8UC1);
  m.put(0, 0, 50.0, 127.0, 220.0);
  System.out.println(m.dump()); // [ 50, 127, 220]
  Mat m2 = new Mat();
  Imgproc.threshold(m, m2, 100, 255, Imgproc.THRESH_BINARY_INV);
  System.out.println(m2.dump()); // [255,   0,   0]
  ```

- `THRESH_TRUNC` — если `src(x, y)` больше `thresh`, то `dst(x, y)` будет иметь значение `thresh`, в противном случае `src(x, y)`. Формат:

  ```
  public static final int THRESH_TRUNC
  ```

  Пример:

  ```
  Mat m = new Mat(1, 3, CvType.CV_8UC1);
  m.put(0, 0, 50.0, 127.0, 220.0);
  System.out.println(m.dump()); // [ 50, 127, 220]
  Mat m2 = new Mat();
  ```

```
Imgproc.threshold(m, m2, 100, 255, Imgproc.THRESH_TRUNC);
System.out.println(m2.dump()); // [ 50, 100, 100]
```

- `THRESH_TOZERO` — если `src(x, y)` больше `thresh`, то `dst(x, y)` будет иметь значение `src(x, y)`, в противном случае `0`. Формат:

  ```
  public static final int THRESH_TOZERO
  ```

  Пример:

  ```
  Mat m = new Mat(1, 3, CvType.CV_8UC1);
  m.put(0, 0, 50.0, 127.0, 220.0);
  System.out.println(m.dump()); // [ 50, 127, 220]
  Mat m2 = new Mat();
  Imgproc.threshold(m, m2, 100, 255, Imgproc.THRESH_TOZERO);
  System.out.println(m2.dump()); // [  0, 127, 220]
  ```

- `THRESH_TOZERO_INV` — если `src(x, y)` больше `thresh`, то `dst(x, y)` будет иметь значение `0`, в противном случае `src(x, y)`. Формат:

  ```
  public static final int THRESH_TOZERO_INV
  ```

  Пример:

  ```
  Mat m = new Mat(1, 3, CvType.CV_8UC1);
  m.put(0, 0, 50.0, 127.0, 220.0);
  System.out.println(m.dump()); // [ 50, 127, 220]
  Mat m2 = new Mat();
  Imgproc.threshold(m, m2, 100, 255, Imgproc.THRESH_TOZERO_INV);
  System.out.println(m2.dump()); // [ 50,   0,   0]
  ```

Выведем значения констант:

```
System.out.println(Imgproc.THRESH_BINARY);     // 0
System.out.println(Imgproc.THRESH_BINARY_INV); // 1
System.out.println(Imgproc.THRESH_TRUNC);      // 2
System.out.println(Imgproc.THRESH_TOZERO);     // 3
System.out.println(Imgproc.THRESH_TOZERO_INV); // 4
```

Результат выполнения метода напрямую зависит от порогового значения. Как правильно подобрать это значение? Если дополнительно после алгоритма указать константу `THRESH_OTSU`, то пороговое значение будет подобрано автоматически. Это значение будет использоваться вместо значения, указанного в параметре `thresh`. Метод вернет подобранное значение. Обратите внимание: константу `THRESH_OTSU` можно указывать только для 8-битного изображения. Формат:

```
public static final int THRESH_OTSU
```

Выведем значение константы:

```
System.out.println(Imgproc.THRESH_OTSU); // 8
```

В версии 3.3 можно также дополнительно указать константу `THRESH_TRIANGLE`. В этом случае пороговое значение будет подобрано автоматически. Обратите вни-

мание: константу `THRESH_TRIANGLE` можно указывать только для 8-битного изображения. Формат:

```
public static final int THRESH_TRIANGLE
```

Выведем значение константы:

```
System.out.println(Imgproc.THRESH_TRIANGLE); // 16
```

Пример преобразования цветного изображения в черно-белое с автоматическим подбором порогового значения приведен в листинге 6.1, а результат выполнения кода из листинга 6.1 показан на рис. 6.1.

Листинг 6.1. Преобразование цветного изображения в черно-белое

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.cvtColor(img, img2, Imgproc.COLOR_BGR2GRAY);
Mat img3 = new Mat();
double thresh = Imgproc.threshold(img2, img3, 100, 255,
                Imgproc.THRESH_BINARY | Imgproc.THRESH_OTSU);
System.out.println(thresh);
CvUtilsFX.showImage(img3, "THRESH_OTSU");
img.release();
img2.release();
img3.release();
```

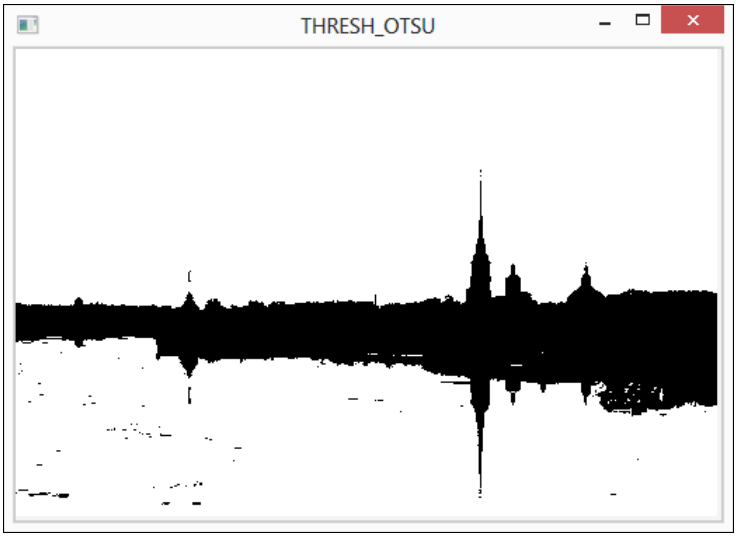


Рис. 6.1. Черно-белое изображение

Для создания черно-белых контуров объектов в 8-битных изображениях можно воспользоваться статическим методом `adaptiveThreshold()` из класса `Imgproc`. Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static void adaptiveThreshold(Mat src, Mat dst, double maxValue,
              int adaptiveMethod, int thresholdType, int blockSize,
              double C)
```

В первом параметре указывается исходное изображение, а во втором параметре — ссылка на матрицу, в которую будет записано измененное изображение. Элементы исходного изображения должны состоять из одного канала и иметь глубину цвета 8 битов. В параметре `maxValue` указывается максимальное значение (не может быть равно нулю). Параметр `adaptiveMethod` задает алгоритм подбора порогового значения (константы `ADAPTIVE_THRESH_MEAN_C` (среднее значение окрестности) или `ADAPTIVE_THRESH_GAUSSIAN_C` (взвешенная сумма значений окрестности) из класса `Imgproc`). Формат:

```
public static final int ADAPTIVE_THRESH_MEAN_C
public static final int ADAPTIVE_THRESH_GAUSSIAN_C
```

Выведем значения констант:

```
System.out.println(Imgproc.ADAPTIVE_THRESH_MEAN_C);     // 0
System.out.println(Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C); // 1
```

В параметре `thresholdType` можно указать алгоритмы `THRESH_BINARY` и `THRESH_BINARY_INV` (инверсия цвета). Параметр `blockSize` задает размер окрестности пиксела (`3`, `5`, `7` и т. д.), а параметр `C` — константу, которая вычитается из среднего или взвешенного значения.

Пример использования метода `adaptiveThreshold()` приведен в листинге 6.2, а результат показан на рис. 6.2.

Листинг 6.2. Метод `adaptiveThreshold()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.cvtColor(img, img2, Imgproc.COLOR_BGR2GRAY);
Mat img3 = new Mat();
Imgproc.adaptiveThreshold(img2, img3, 255,
                          Imgproc.ADAPTIVE_THRESH_MEAN_C,
                          Imgproc.THRESH_BINARY, 3, 5);
Mat img4 = new Mat();
Imgproc.adaptiveThreshold(img2, img4, 255,
                          Imgproc.ADAPTIVE_THRESH_GAUSSIAN_C,
                          Imgproc.THRESH_BINARY_INV, 3, 5);
```

```
CvUtilsFX.showImage(img3, "ADAPTIVE_THRESH_MEAN_C");
CvUtilsFX.showImage(img4,
                    "ADAPTIVE_THRESH_GAUSSIAN_C + THRESH_BINARY_INV");

// Инверсия с помощью таблицы соответствия
Mat lut = new Mat(1, 256, CvType.CV_8UC1);
byte[] arr = new byte[256];
for (int i = 0; i < 256; i++) {
   arr[i] = (byte) (255 - i);
}
lut.put(0, 0, arr);
// Преобразование в соответствии с таблицей
Mat imgInv = new Mat();
Core.LUT(img3, lut, imgInv);
CvUtilsFX.showImage(imgInv, "ADAPTIVE_THRESH_MEAN_C + Inv");
img.release();      img2.release();
img3.release();     img4.release();
imgInv.release();   lut.release();
```

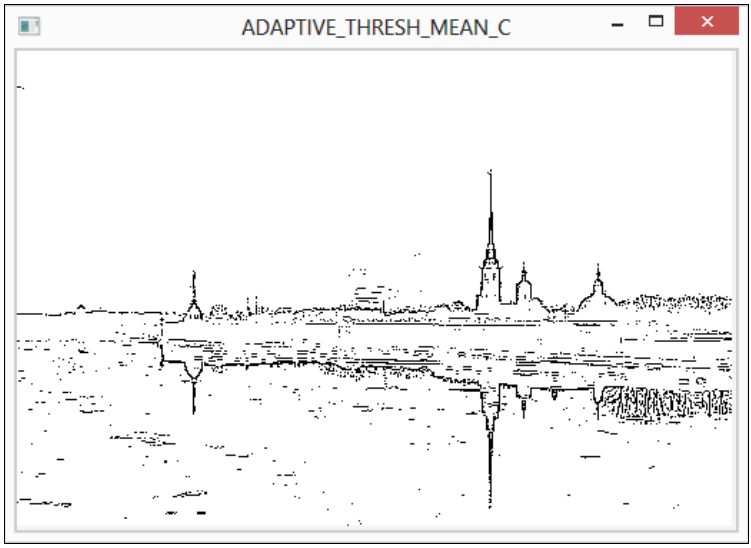


Рис. 6.2. Результат применения метода `adaptiveThreshold()`

Примечание

Преобразовать изображение в черно-белое можно также с помощью статического метода `compare()` из класса `Core` (см. *разд. 2.7*).

6.2. Изменение яркости и насыщенности

Яркость и насыщенность являются отдельными каналами в цветовой модели HSV (HSB). Поэтому для изменения их значений достаточно преобразовать модель BGR в HSV, изменить значения соответствующего канала, а затем преобразовать модель HSV в BGR.

Пример изменения яркости приведен в листинге 6.3, а пример изменения насыщенности — в листинге 6.4.

Листинг 6.3. Изменение яркости

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat imgHSV = new Mat();
Imgproc.cvtColor(img, imgHSV, Imgproc.COLOR_BGR2HSV);
// Увеличение яркости
Core.add(imgHSV, new Scalar(0, 0, 40), imgHSV);
// Уменьшение яркости
//Core.add(imgHSV, new Scalar(0, 0, -40), imgHSV);
Mat imgBGR = new Mat();
Imgproc.cvtColor(imgHSV, imgBGR, Imgproc.COLOR_HSV2BGR);
CvUtilsFX.showImage(img, "Яркость +0");
CvUtilsFX.showImage(imgBGR, "Яркость +40");
img.release();
imgHSV.release();
imgBGR.release();
```

Листинг 6.4. Изменение насыщенности

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat imgHSV = new Mat();
Imgproc.cvtColor(img, imgHSV, Imgproc.COLOR_BGR2HSV);
// Увеличение насыщенности
Core.add(imgHSV, new Scalar(0, 40, 0), imgHSV);
// Уменьшение насыщенности
//Core.add(imgHSV, new Scalar(0, -40, 0), imgHSV);
Mat imgBGR = new Mat();
Imgproc.cvtColor(imgHSV, imgBGR, Imgproc.COLOR_HSV2BGR);
```

```
CvUtilsFX.showImage(img, "Насыщенность +0");
CvUtilsFX.showImage(imgBGR, "Насыщенность +40");
img.release();
imgHSV.release();
imgBGR.release();
```

Если обнулить значение канала `S` (насыщенность) в цветовой модели `HSV`, то изображение будет в оттенках серого:

```
// В оттенки серого
Core.add(imgHSV, new Scalar(0, -255, 0), imgHSV);
```

В цветовой модели `Lab` есть отдельный канал `L` (Lightness, светлота), который также можно использовать для изменения яркости изображения (листинг 6.5).

Листинг 6.5. Изменение светлоты в цветовой модели `Lab`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat imgLab = new Mat();
Imgproc.cvtColor(img, imgLab, Imgproc.COLOR_BGR2Lab);
// Увеличение светлоты
Core.add(imgLab, new Scalar(40, 0, 0), imgLab);
// Уменьшение светлоты
//Core.add(imgLab, new Scalar(-40, 0, 0), imgLab);
Mat imgBGR = new Mat();
Imgproc.cvtColor(imgLab, imgBGR, Imgproc.COLOR_Lab2BGR);
CvUtilsFX.showImage(img, "Светлота +0");
CvUtilsFX.showImage(imgBGR, "Светлота +40");
img.release();
imgLab.release();
imgBGR.release();
```

6.3. Изменение цветового баланса

В цветовой модели `Lab` канал светлоты (яркости) отделен от двух каналов цвета:

- `a` — положение цвета от зеленого до красного;
- `b` — положение цвета от синего до желтого.

Меняя значения этих двух каналов, можно изменять цветовой баланс изображения. Для изменения значений необходимо преобразовать модель `BGR` в `Lab`, изменить значения соответствующего канала, а затем преобразовать модель `Lab` в `BGR`. Пример изменения цветового баланса приведен в листинге 6.6.

Листинг 6.6. Изменение цветового баланса

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat imgLab = new Mat();
Imgproc.cvtColor(img, imgLab, Imgproc.COLOR_BGR2Lab);
// Положение цвета от зеленого до красного
//Core.add(imgLab, new Scalar(0, 10, 0), imgLab);
// Положение цвета от синего до желтого
Core.add(imgLab, new Scalar(0, 0, 20), imgLab);
Mat imgBGR = new Mat();
Imgproc.cvtColor(imgLab, imgBGR, Imgproc.COLOR_Lab2BGR);
CvUtilsFX.showImage(img, "+0");
CvUtilsFX.showImage(imgBGR, "+N");
img.release();
imgLab.release();
imgBGR.release();
```

6.4. Изменение контраста

В библиотеке OpenCV нет специализированного метода для изменения контраста изображения, поэтому попробуем реализовать его самостоятельно. За основу возьмем телевизионный алгоритм изменения контраста для цветовой модели RGB. Реализация алгоритма состоит из нескольких этапов:

1. Вычисление средней яркости изображения.
2. Построение таблицы соответствия.
3. Применение таблицы соответствия к изображению.

Вычислить среднюю яркость изображения можно, например, следующим образом:

```
int ch = img.channels();
byte[] ar = new byte[img.cols() * img.rows() * ch];
img.get(0, 0, ar);
double mean = 0;
for (int i = 0, j = ar.length; i < j; i+=ch) {
   mean += ((ar[i] & 0xFF) * 0.114 +
            (ar[i + 1] & 0xFF) * 0.587 +
            (ar[i + 2] & 0xFF) * 0.299);
}
mean = mean / (img.cols() * img.rows());
```

Обратите внимание: для каждого цвета существует свой коэффициент, показывающий восприятие яркости цвета человеком (`0.114` — для синего цвета, `0.587` — для зеленого и `0.299` для красного).

Этот код будет работать для изображений, имеющих три или четыре канала, но медленно, т. к. он содержит несколько лишних действий. Для ускорения кода воспользуемся статическим методом `mean()`, возвращающим средние значения для каждого канала изображения:

```
Scalar meanBGR = Core.mean(img);
double mean = meanBGR.val[0] * 0.114 + meanBGR.val[1] * 0.587 +
              meanBGR.val[2] * 0.299;
```

Далее для каждого значения мы должны создать таблицу соответствия, используя следующую формулу:

```
color = (int) (contrast * (i - mean) + mean);
```

где `contrast` — коэффициент контраста, `i` — текущее значение, а `mean` — средняя яркость изображения. Если коэффициент равен `1`, то изображение не меняется, значение больше `1` — увеличивает контраст, а значение меньше `1` — уменьшает его. Можно также воспользоваться следующей формулой (средняя яркость не учитывается):

```
color = (int) ((contrast * (i / 255.0 - 0.5) + 0.5) * 255);
```

Или просто указать значения средней яркости вручную:

```
color = (int) (contrast * (i - 128) + 128);
```

После вычисления значения цвета необходимо выполнить проверку выхода значения за границы диапазона и обязательно произвести нормализацию:

```
color = color > 255 ? 255 : (color < 0 ? 0 : color);
```

На последнем этапе применяем таблицу соответствия к изображению, используя статический метод `LUT()` из класса `Core`:

```
Mat img2 = new Mat();
Core.LUT(img, lut, img2);
```

Полный код примера приведен в листинге 6.7.

Листинг 6.7. Изменение контраста

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
// Вычисление средней яркости изображения
Scalar meanBGR = Core.mean(img);
double mean = meanBGR.val[0] * 0.114 + meanBGR.val[1] * 0.587 +
              meanBGR.val[2] * 0.299;
// Коэффициент контраста
double contrast = 1.5;
```

```
// Построение таблицы соответствия
Mat lut = new Mat(1, 256, CvType.CV_8UC1);
byte[] arr = new byte[256];
int color = 0;
for (int i = 0; i < 256; i++) {
   color = (int) (contrast * (i - mean) + mean);
   //color = (int) (contrast * (i - 128) + 128);
   //color = (int) ((contrast * (i / 255.0 - 0.5) + 0.5) * 255);
   color = color > 255 ? 255 : (color < 0 ? 0 : color);
   arr[i] = (byte) (color);
}
lut.put(0, 0, arr);
// Применение таблицы соответствия к изображению
Mat img2 = new Mat();
Core.LUT(img, lut, img2);
CvUtilsFX.showImage(img2, "Контраст: " + contrast);
img.release();
img2.release();
lut.release();
```

6.5. Создание негатива изображения

Чтобы создать негатив изображения, нужно из значения `255` вычесть текущее значение компонента цвета пиксела. Для этого можно воспользоваться статическим методом `subtract()` из класса `Core`, предварительно создав и заполнив матрицу значениями `255` (листинг 6.8).

Листинг 6.8. Создание негатива изображения

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat m = new Mat(img.rows(), img.cols(), img.type(), CvUtils.COLOR_WHITE);
Mat negative = new Mat();
Core.subtract(m, img, negative);
CvUtilsFX.showImage(negative, "Негатив");
img.release();
negative.release();
m.release();
```

6.6. Сепия

Для создания эффекта сепии (состаривания фотографии) нужно умножить все компоненты цвета на цвет сепии и усреднить значения. Если использовать для преобразования массив, то код будет выглядеть следующим образом:

```
int ch = img.channels();
byte[] arr = new byte[img.cols() * img.rows() * ch];
img.get(0, 0, arr);
int red = 0, green = 0, blue = 0, r = 0, g = 0, b = 0;
for (int i = 0, j = arr.length; i < j; i+=ch) {
   b = arr[i]     & 0xFF;
   g = arr[i + 1] & 0xFF;
   r = arr[i + 2] & 0xFF;
   blue  = (int) (r * 0.272 + g * 0.534 + b * 0.131);
   green = (int) (r * 0.349 + g * 0.686 + b * 0.168);
   red   = (int) (r * 0.393 + g * 0.769 + b * 0.189);
   blue  = blue > 255 ? 255 : (blue < 0 ? 0 : blue);
   green = green > 255 ? 255 : (green < 0 ? 0 : green);
   red   = red > 255 ? 255 : (red < 0 ? 0 : red);
   arr[i]     = (byte) blue;
   arr[i + 1] = (byte) green;
   arr[i + 2] = (byte) red;
}
img.put(0, 0, arr);
```

Этот код опять-таки выполняется достаточно медленно. Для ускорения можно воспользоваться статическим методом `transform()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void transform(Mat src, Mat dst, Mat m)
```

В первом параметре указывается исходное изображение, во втором — матрица, в которую будет записан результат, а в третьем — матрица трансформации (с вещественными значениями — например, матрица типа `CV_32F`).

Пример использования метода `transform()` для создания эффекта сепии приведен в листинге 6.9.

Листинг 6.9. Сепия

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
// Построение матрицы трансформации
Mat kernel = new Mat(3, 3, CvType.CV_32FC1);
kernel.put(0, 0,
      0.131, 0.534, 0.272, // blue  = b * b1 + g * g1 + r * r1
```

```
      0.168, 0.686, 0.349, // green = b * b2 + g * g2 + r * r2
      0.189, 0.769, 0.393  // red   = b * b3 + g * g3 + r * r3
);
Mat sepia = new Mat();
Core.transform(img, sepia, kernel);
CvUtilsFX.showImage(sepia, "Сепия");
img.release();
kernel.release();
sepia.release();
```

6.7. Вычисление гистограммы

Как уже было отмечено ранее, *гистограмма* — это статистическая информация о количестве пикселов, имеющих одинаковое значение. Если мы сфотографируем пейзаж днем в пасмурную погоду и вычислим гистограмму по изображению в оттенках серого, то при правильно выбранной экспозиции получим плавное возрастание значений до середины гистограммы, а затем их плавное уменьшение. Для снимка, сделанного ночью, пик сместится ближе к левой границе гистограммы, т. к. темных пикселов в нем будет больше, чем светлых. Если же мы сделаем фотографию зимой днем, то пик сместится ближе к ее правой границе, т. к. снег описывается пикселами, имеющими светлые значения, и этих пикселов будет много.

Предыдущие примеры основаны на предположении, что при съемке была выбрана правильная экспозиция. Если предполагать, что съемка выполняется днем, то по гистограмме можно определить правильность выбора экспозиции: когда пик смещен ближе к ее левой границе, то снимок слишком темный, а когда он ближе к правой — то слишком светлый. Если имеются пикселы с крайними значениями, значит при съемке была потеряна информация в темных областях (обычно в тени при солнечной погоде) или в светлых областях (например, в кадр попало солнце, или имеется блик на отражающей поверхности). С помощью гистограммы можно также определить, является ли изображение контрастным: если пик расположен в середине, но возрастание значений начинается не от границы, изображение может быть бледным. При этом, чем ближе начало возрастания значений к середине гистограммы, тем менее контрастно изображение.

Таким образом, определив по гистограмме правильность экспонирования изображения, мы можем при необходимости выполнить его обработку, наблюдая за ее изменениями. Вычислить гистограмму позволяет статический метод `calcHist()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void calcHist(List<Mat> images, MatOfInt channels,
                        Mat mask, Mat hist, MatOfInt histSize,
                        MatOfFloat ranges)
public static void calcHist(List<Mat> images, MatOfInt channels,
                        Mat mask, Mat hist, MatOfInt histSize,
                        MatOfFloat ranges, boolean accumulate)
```

В первом параметре указывается список с изображениями (глубина `CV_8U` или `CV_32F`). Параметр `channels` задает индексы каналов внутри списка `images`, по которым будет вычисляться гистограмма. В параметре `mask` можно указать маску. Результат вычисления гистограммы будет записан в матрицу, указанную в параметре `hist`. Параметр `histSize` задает размер гистограммы, а параметр `ranges` — диапазон значений гистограммы. Если в параметре `accumulate` указать значение `true`, то вычисленные значения будут прибавляться к уже имеющимся значениям в матрице `hist`.

Пример вычисления гистограммы и ее отрисовки приведен в листинге 6.10.

Листинг 6.10. Вычисление гистограммы

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");
Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
// Вычисляем гистограммы по каналам
ArrayList<Mat> images = new ArrayList<Mat>();
images.add(img);
images.add(imgGray);
Mat histGray = new Mat();
Mat histRed = new Mat();
Mat histGreen = new Mat();
Mat histBlue = new Mat();
Imgproc.calcHist(images, new MatOfInt(3), new Mat(),
        histGray, new MatOfInt(256), new MatOfFloat(0, 256));
Imgproc.calcHist(images, new MatOfInt(2), new Mat(),
        histRed, new MatOfInt(256), new MatOfFloat(0, 256));
Imgproc.calcHist(images, new MatOfInt(1), new Mat(),
        histGreen, new MatOfInt(256), new MatOfFloat(0, 256));
Imgproc.calcHist(images, new MatOfInt(0), new Mat(),
        histBlue, new MatOfInt(256), new MatOfFloat(0, 256));
// Нормализация диапазона
Core.normalize(histGray, histGray, 0, 128, Core.NORM_MINMAX);
Core.normalize(histRed, histRed, 0, 128, Core.NORM_MINMAX);
Core.normalize(histGreen, histGreen, 0, 128, Core.NORM_MINMAX);
Core.normalize(histBlue, histBlue, 0, 128, Core.NORM_MINMAX);
// Отрисовка гистограмм
double v = 0;
int h = 150;
Mat imgHistRed = new Mat(h, 256, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Mat imgHistGreen = new Mat(h, 256, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
```

```
Mat imgHistBlue = new Mat(h, 256, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Mat imgHistGray = new Mat(h, 256, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
for (int i = 0, j = histGray.rows(); i < j; i++) {
   v = Math.round(histRed.get(i, 0)[0]);
   if (v != 0) {
      Imgproc.line(imgHistRed, new Point(i, h - 1),
                   new Point(i, h - 1 - v), CvUtils.COLOR_RED);
   }
   v = Math.round(histGreen.get(i, 0)[0]);
   if (v != 0) {
      Imgproc.line(imgHistGreen, new Point(i, h - 1),
                   new Point(i, h - 1 - v), CvUtils.COLOR_GREEN);
   }
   v = Math.round(histBlue.get(i, 0)[0]);
   if (v != 0) {
      Imgproc.line(imgHistBlue, new Point(i, h - 1),
                   new Point(i, h - 1 - v), CvUtils.COLOR_BLUE);
   }
   v = Math.round(histGray.get(i, 0)[0]);
   if (v != 0) {
      Imgproc.line(imgHistGray, new Point(i, h - 1),
                   new Point(i, h - 1 - v), CvUtils.COLOR_GRAY);
   }
}
CvUtilsFX.showImage(imgHistRed, "Red");
CvUtilsFX.showImage(imgHistGreen, "Green");
CvUtilsFX.showImage(imgHistBlue, "Blue");
CvUtilsFX.showImage(imgHistGray, "Gray");
img.release();          imgGray.release();
imgHistRed.release();   imgHistGreen.release();
imgHistBlue.release();  imgHistGray.release();
histGray.release();     histRed.release();
histGreen.release();    histBlue.release();
```

Выполнить автоматическое выравнивание гистограммы для 8-битного изображения в градациях серого можно с помощью метода `equalizeHist()` (см. *разд. 6.8*) или класса `CLAHE` (см. *разд. 6.9*). Сравнить две гистограммы позволяет статический метод `compareHist()` из класса `Imgproc`. Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static double compareHist(Mat H1, Mat H2, int method)
```

В параметре `method` указывается способ сравнения. Можно указать следующие константы из класса `Imgproc`:

```
public static final int HISTCMP_CORREL
public static final int HISTCMP_CHISQR
```

```
public static final int HISTCMP_INTERSECT
public static final int HISTCMP_BHATTACHARYYA
public static final int HISTCMP_HELLINGER
public static final int HISTCMP_CHISQR_ALT
public static final int HISTCMP_KL_DIV
```

В версии 2.4 используются следующие константы:

```
public static final int CV_COMP_CORREL
public static final int CV_COMP_CHISQR
public static final int CV_COMP_INTERSECT
public static final int CV_COMP_BHATTACHARYYA
public static final int CV_COMP_HELLINGER
```

Выведем значения констант:

```
System.out.println(Imgproc.HISTCMP_CORREL);        // 0
System.out.println(Imgproc.HISTCMP_CHISQR);        // 1
System.out.println(Imgproc.HISTCMP_INTERSECT);     // 2
System.out.println(Imgproc.HISTCMP_BHATTACHARYYA); // 3
System.out.println(Imgproc.HISTCMP_HELLINGER);     // 3
System.out.println(Imgproc.HISTCMP_CHISQR_ALT);    // 4
System.out.println(Imgproc.HISTCMP_KL_DIV);        // 5
```

При сравнении гистограмм следует учитывать, что гистограмма дает представление о количестве пикселов с одинаковой яркостью, но не дает никакого представления, где эти пикселы находятся внутри изображения. У абсолютно разных изображений могут быть одинаковые гистограммы.

6.8. Автоматическое выравнивание гистограммы изображения в градациях серого

С помощью статического метода `equalizeHist()` из класса `Imgproc` можно выполнить автоматическую нормализацию яркости и контраста 8-битного изображения в градациях серого. Формат метода:

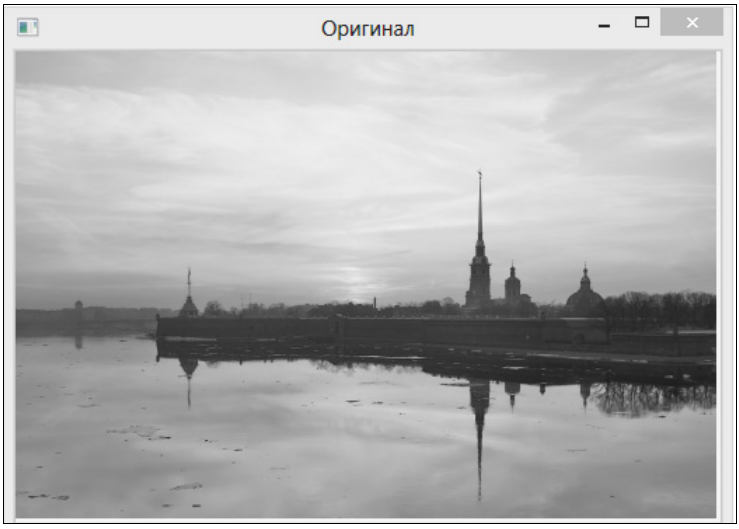
```
import org.opencv.imgproc.Imgproc;
public static void equalizeHist(Mat src, Mat dst)
```

В первом параметре указывается исходное изображение, а во втором параметре — ссылка на матрицу, в которую будет записано измененное изображение. Элементы исходного изображения должны состоять из одного канала и иметь глубину цвета 8 битов.

Пример использования метода `equalizeHist()` приведен в листинге 6.11. Результат выполнения кода из листинга 6.11 показан на рис. 6.3, а гистограммы до и после операции можно увидеть на рис. 6.4.

Листинг 6.11. Метод `equalizeHist()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.cvtColor(img, img2, Imgproc.COLOR_BGR2GRAY);
Mat img3 = new Mat();
Imgproc.equalizeHist(img2, img3);
// Вычисляем и отрисовываем гистограммы
ArrayList<Mat> images = new ArrayList<Mat>();
images.add(img2);
images.add(img3);
Mat hist = new Mat();
Mat hist2 = new Mat();
Imgproc.calcHist(images, new MatOfInt(0), new Mat(),
                 hist, new MatOfInt(256), new MatOfFloat(0, 256));
Imgproc.calcHist(images, new MatOfInt(1), new Mat(),
                 hist2, new MatOfInt(256), new MatOfFloat(0, 256));
Core.normalize(hist, hist, 0, 128, Core.NORM_MINMAX);
Core.normalize(hist2, hist2, 0, 128, Core.NORM_MINMAX);
double v = 0;
int h = 150;
Mat imgHist = new Mat(h, 256, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Mat imgHist2 = new Mat(h, 256, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
for (int i = 0, j = hist.rows(); i < j; i++) {
   v = Math.round(hist.get(i, 0)[0]);
   if (v != 0) {
      Imgproc.line(imgHist, new Point(i, h - 1),
                   new Point(i, h - 1 - v), CvUtils.COLOR_BLACK);
   }
   v = Math.round(hist2.get(i, 0)[0]);
   if (v != 0) {
      Imgproc.line(imgHist2, new Point(i, h - 1),
                   new Point(i, h - 1 - v), CvUtils.COLOR_BLACK);
   }
}
CvUtilsFX.showImage(img2, "Оригинал");
CvUtilsFX.showImage(imgHist, "Гистограмма до");
CvUtilsFX.showImage(img3, "equalizeHist");
CvUtilsFX.showImage(imgHist2, "Гистограмма после");
img.release();     img2.release();    img3.release();
imgHist.release(); imgHist2.release();
hist.release();    hist2.release();
```



а

б

Рис. 6.3. Результат выполнения кода из листинга 6.11: *а* — до обработки; *б* — после обработки

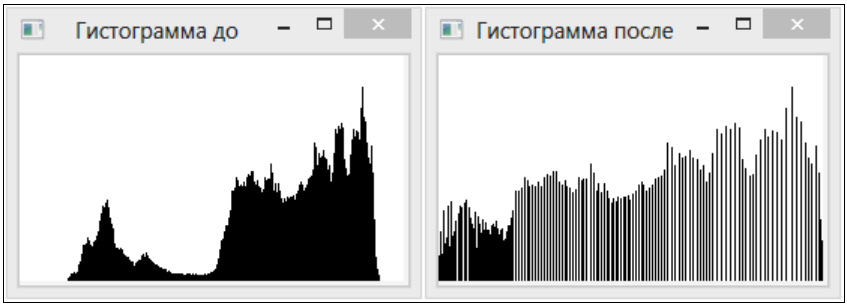


Рис. 6.4. Гистограммы до (*слева*) и после (*справа*) применения метода `equalizeHist()`

6.9. Класс *CLAHE*

Класс `CLAHE` выполняет адаптивное выравнивание гистограммы с помощью алгоритма Contrast Limited Adaptive Histogram Equalization. Инструкция импорта:

```
import org.opencv.imgproc.CLAHE;
```

Иерархия наследования:

```
Object — Algorithm — CLAHE
```

Создать объект позволяет статический метод `createCLAHE()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static CLAHE createCLAHE()
public static CLAHE createCLAHE(double clipLimit, Size tileGridSize)
```

Изменить значение параметра `clipLimit` (значение по умолчанию: `40.0`) после создания объекта позволяет метод `setClipLimit()`. Формат метода:

```
public void setClipLimit(double clipLimit)
```

Изменить значение параметра `tileGridSize` (значение по умолчанию: `Size(8, 8)`) после создания объекта позволяет метод `setTilesGridSize()`. Формат метода:

```
public void setTilesGridSize(Size tileGridSize)
```

Чтобы применить алгоритм к изображению, следует вызвать метод `apply()`. Формат метода:

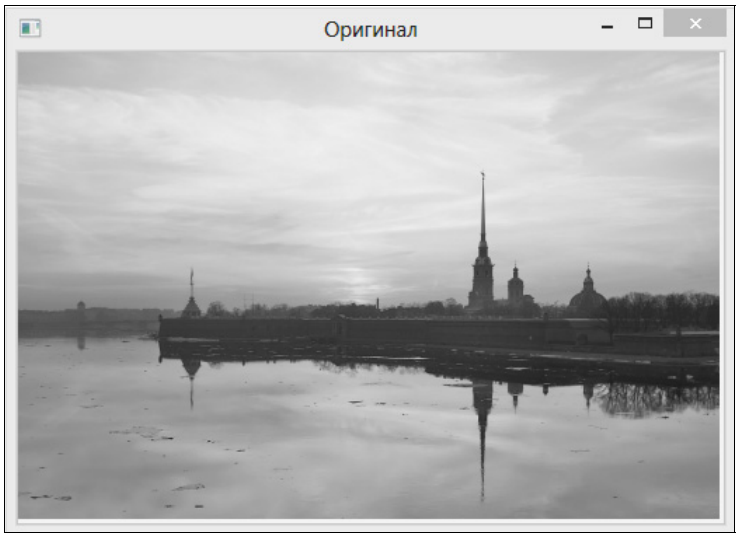
```
public void apply(Mat src, Mat dst)
```

В первом параметре указывается исходное изображение (тип `CV_8UC1` или `CV_16UC1`), а во втором — матрица, в которую будет записан результат операции.

Пример использования класса `CLAHE` приведен в листинге 6.12. Результат выполнения кода из листинга 6.12 показан на рис. 6.5, а гистограммы до и после операции можно увидеть на рис. 6.6.

Листинг 6.12. Класс `CLAHE`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.cvtColor(img, img2, Imgproc.COLOR_BGR2GRAY);
Mat img3 = new Mat();
CLAHE clane = Imgproc.createCLAHE();
clane.setClipLimit(4);
clane.apply(img2, img3);
```



а

б

Рис. 6.5. Результат выполнения кода из листинга 6.12: *а* — до обработки; *б* — после обработки

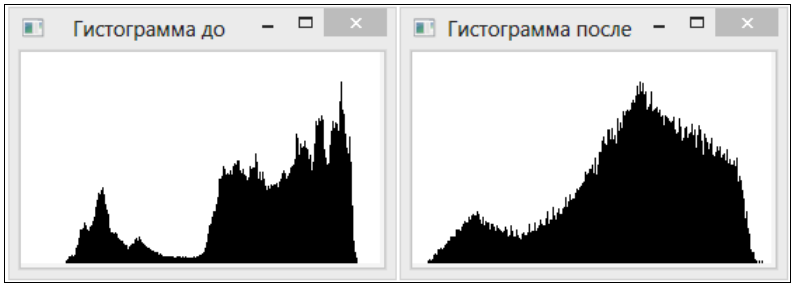


Рис. 6.6. Гистограммы до (*слева*) и после (*справа*) использования класса `CLAHE`

```
// Вычисляем и отрисовываем гистограммы
ArrayList<Mat> images = new ArrayList<Mat>();
images.add(img2);
images.add(img3);
Mat hist = new Mat();
Mat hist2 = new Mat();
Imgproc.calcHist(images, new MatOfInt(0), new Mat(),
                 hist, new MatOfInt(256), new MatOfFloat(0, 256));
Imgproc.calcHist(images, new MatOfInt(1), new Mat(),
                 hist2, new MatOfInt(256), new MatOfFloat(0, 256));
Core.normalize(hist, hist, 0, 128, Core.NORM_MINMAX);
Core.normalize(hist2, hist2, 0, 128, Core.NORM_MINMAX);
double v = 0;
int h = 150;
Mat imgHist = new Mat(h, 256, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Mat imgHist2 = new Mat(h, 256, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
for (int i = 0, j = hist.rows(); i < j; i++) {
   v = Math.round(hist.get(i, 0)[0]);
   if (v != 0) {
      Imgproc.line(imgHist, new Point(i, h - 1),
                   new Point(i, h - 1 - v), CvUtils.COLOR_BLACK);
   }
   v = Math.round(hist2.get(i, 0)[0]);
   if (v != 0) {
      Imgproc.line(imgHist2, new Point(i, h - 1),
                   new Point(i, h - 1 - v), CvUtils.COLOR_BLACK);
   }
}
CvUtilsFX.showImage(img2, "Оригинал");
CvUtilsFX.showImage(imgHist, "Гистограмма до");
CvUtilsFX.showImage(img3, "CLAHE");
CvUtilsFX.showImage(imgHist2, "Гистограмма после");
img.release();     img2.release();    img3.release();
imgHist.release(); imgHist2.release();
hist.release();    hist2.release();
```

6.10. Метод *applyColorMap()*

Статический метод `applyColorMap()` из класса `Imgproc` позволяет применить к изображению различные цветовые палитры. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void applyColorMap(Mat src, Mat dst, int colormap)
public static void applyColorMap(Mat src, Mat dst, Mat userColor)
```

В первом параметре указывается исходное изображение (тип `CV_8UC1` или `CV_8UC3`), а во втором — ссылка на матрицу, в которую будет записан результат операции. В параметре `colormap` можно указать следующие константы из класса `Imgproc`:

```
public static final int COLORMAP_AUTUMN
public static final int COLORMAP_BONE
public static final int COLORMAP_JET
public static final int COLORMAP_WINTER
public static final int COLORMAP_RAINBOW
public static final int COLORMAP_OCEAN
public static final int COLORMAP_SUMMER
public static final int COLORMAP_SPRING
public static final int COLORMAP_COOL
public static final int COLORMAP_HSV
public static final int COLORMAP_PINK
public static final int COLORMAP_HOT
public static final int COLORMAP_PARULA
```

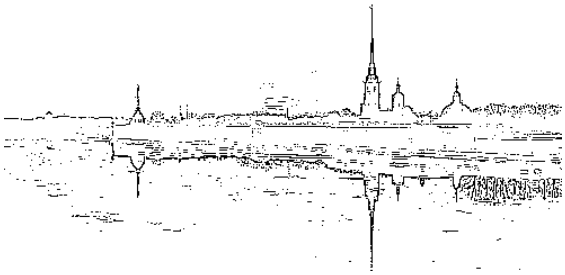
Пример использования метода `applyColorMap()` приведен в листинге 6.13.

Листинг 6.13. Метод `applyColorMap()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
System.out.println(img.size());
Imgproc.resize(img, img, new Size(), 0.6, 0.6, Imgproc.INTER_LINEAR);
CvUtilsFX.showImage(img, "Оригинал");

Mat img2 = new Mat();
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_AUTUMN);
CvUtilsFX.showImage(img2, "COLORMAP_AUTUMN");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_BONE);
CvUtilsFX.showImage(img2, "COLORMAP_BONE");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_COOL);
CvUtilsFX.showImage(img2, "COLORMAP_COOL");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_HOT);
CvUtilsFX.showImage(img2, "COLORMAP_HOT");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_HSV);
CvUtilsFX.showImage(img2, "COLORMAP_HSV");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_JET);
CvUtilsFX.showImage(img2, "COLORMAP_JET");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_OCEAN);
CvUtilsFX.showImage(img2, "COLORMAP_OCEAN");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_PARULA);
CvUtilsFX.showImage(img2, "COLORMAP_PARULA");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_PINK);
CvUtilsFX.showImage(img2, "COLORMAP_PINK");
```

```
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_RAINBOW);
CvUtilsFX.showImage(img2, "COLORMAP_RAINBOW");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_SPRING);
CvUtilsFX.showImage(img2, "COLORMAP_SPRING");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_SUMMER);
CvUtilsFX.showImage(img2, "COLORMAP_SUMMER");
Imgproc.applyColorMap(img, img2, Imgproc.COLORMAP_WINTER);
CvUtilsFX.showImage(img2, "COLORMAP_WINTER");
img.release();
img2.release();
```

ГЛАВА 7



Применение фильтров

Применение фильтров к изображению позволяет вычислить свертку, медиану или выполнить морфологическую операцию (расширение или сжатие). Ядро фильтра можно представить в виде окна квадратной или прямоугольной формы. Чаще всего используются квадратные ядра с размерами `3x3`, `5x5`, `7x7` и `9x9`. Внутри этого окна существует опорная точка, которая чаще всего располагается в середине окна.

Вначале окно помещают в самое начало изображения и совмещают опорную точку с первым пикселом. Потом производится вычисление значений внутри окна — например, вычисление свертки, а затем результат присваивается пикселу, совпадающему с положением опорной точки. Далее окно сдвигается к следующему пикселу. Операция повторяется, пока не будет достигнут конец изображения.

Если окно совместить с пикселом, расположенным около границы, то можно заметить, что части окна окажутся за пределами изображения, и неоткуда будет брать значения для вычисления. В этом случае нужно сгенерировать недостающие пикселы, создав вокруг изображения рамку. В *разд. 4.9* при изучении метода `copyMakeBorder()` мы уже рассматривали возможность создания рамки. Только для генерации недостающих пикселов при использовании фильтров применяется не метод `copyMakeBorder()`, а метод `borderInterpolate()`. Формат метода:

```
import org.opencv.core.Core;
public static int borderInterpolate(int p, int len, int borderType)
```

Параметр `borderType` задает тип рамки. В качестве значения указываются те же самые константы, что и при использовании метода `copyMakeBorder()`. Работать напрямую с методом `borderInterpolate()` нам не придется, а вот типы рамок нужно будет помнить. В большинстве случаев тип по умолчанию `BORDER_DEFAULT`, соответствующий типу `BORDER_REFLECT_101`, подходит лучше всего.

7.1. Размытие и подавление цифрового шума

Размытие используется для сглаживания изображения — например, для устранения шума, возникающего при съемке с длительными выдержками или с высокими значениями ISO. Если шум не подавить, то при распознавании имеющихся на изобра-

жении образов могут возникнуть проблемы. Размыть изображение можно с помощью методов `boxFilter()` и `blur()`, двустороннего фильтра, фильтра Гаусса (наиболее часто используемый способ размытия) или медианного фильтра.

7.1.1. Метод *blur()*: однородное сглаживание

Форматы метода `blur()`:

```
import org.opencv.imgproc.Imgproc;
public static void blur(Mat src, Mat dst, Size ksize)
public static void blur(Mat src, Mat dst, Size ksize, Point anchor)
public static void blur(Mat src, Mat dst, Size ksize, Point anchor,
                        int borderType)
```

В первом параметре указывается исходное изображение (глубина `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` или `CV_64F`), а во втором — матрица, в которую будет записан результат операции. В параметре `ksize` указываются размеры ядра фильтра: чем больше размеры ядра, тем больше размытие. Параметр `anchor` задает положение опорной точки. По умолчанию используется значение `Point(-1, -1)`, которое означает, что опорная точка находится в центре ядра. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`.

Пример однородного сглаживания по методу `blur()` приведен в листинге 7.1.

Листинг 7.1. Метод `blur()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.blur(img, img2, new Size(3, 3));
CvUtilsFX.showImage(img2, "Size(3, 3)");
Mat img3 = new Mat();
Imgproc.blur(img, img3, new Size(45, 45), new Point(-1, -1));
CvUtilsFX.showImage(img3, "Size(45, 45)");
img.release();
img2.release();
img3.release();
```

Значения элементов матрицы ядра, при использовании метода `blur()`, рассчитываются следующим образом:

```
Mat kernel = Mat.ones(new Size(3, 3), CvType.CV_32FC1);
Core.divide(kernel, new Scalar(kernel.rows() * kernel.cols()),
            kernel);
```

```
System.out.println(kernel.dump());
/*
[0.11111111, 0.11111111, 0.11111111;
 0.11111111, 0.11111111, 0.11111111;
 0.11111111, 0.11111111, 0.11111111]*/
```

7.1.2. Размытие по Гауссу

Размытие по Гауссу реализуется с помощью статического метода `GaussianBlur()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void GaussianBlur(Mat src, Mat dst, Size ksize,
                          double sigmaX)
public static void GaussianBlur(Mat src, Mat dst, Size ksize,
                          double sigmaX, double sigmaY)
public static void GaussianBlur(Mat src, Mat dst, Size ksize,
                          double sigmaX, double sigmaY, int borderType)
```

В первом параметре указывается исходное изображение (глубина `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` или `CV_64F`), а во втором — матрица, в которую будет записан результат операции. В параметре `ksize` указываются размеры ядра фильтра. Ширина и высота ядра могут различаться, но оба они должны быть положительными и нечетными. Если размер равен `0`, то он рассчитывается из параметров `sigmaX` и `sigmaY`. Чем больше размеры ядра, тем больше размытие. Параметр `sigmaX` задает стандартное отклонение по оси `X`, а параметр `sigmaY` — по оси `Y`. Если параметр `sigmaY` равен `0`, то его значение будет равно значению параметра `sigmaX`. Если оба параметра равны `0`, то их значения вычисляются из размеров ядра. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`.

Пример размытия по методу `GaussianBlur()` приведен в листинге 7.2.

Листинг 7.2. Метод `GaussianBlur()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.GaussianBlur(img, img2, new Size(3, 3), 0);
CvUtilsFX.showImage(img2, "Size(3, 3)");
Mat img3 = new Mat();
Imgproc.GaussianBlur(img, img3, new Size(45, 45), 0);
CvUtilsFX.showImage(img3, "Size(45, 45)");
Mat img4 = new Mat();
Imgproc.GaussianBlur(img, img4, new Size(0, 0), 1.5);
```

```
CvUtilsFX.showImage(img4, "Size(0, 0), 1.5");
img.release();
img2.release();
img3.release();
img4.release();
```

Получить коэффициенты, используемые фильтром Гаусса, позволяет статический метод `getGaussianKernel()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static Mat getGaussianKernel(int ksize, double sigma)
public static Mat getGaussianKernel(int ksize, double sigma, int ktype)
```

В первом параметре указывается размер ядра фильтра (положительное нечетное число), а во втором параметре — стандартное отклонение. В параметре `ktype` можно указать значение `CV_32F` или `CV_64F`.

Пример:

```
Mat kernel = Imgproc.getGaussianKernel(3, 1.5, CvType.CV_64F);
System.out.println(kernel.dump());
/*
[0.30780132912347;
 0.38439734175306;
 0.30780132912347]*/
```

7.1.3. Метод *bilateralFilter()*: двустороннее сглаживание

Большинство фильтров не только удаляют шум, но и сглаживают границы объектов. Статический метод `bilateralFilter()` из класса `Imgproc` старается сохранить границы объектов, воздействуя в основном на шум. Недостатком фильтра является более медленная его работа. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void bilateralFilter(Mat src, Mat dst, int d,
                           double sigmaColor, double sigmaSpace)
public static void bilateralFilter(Mat src, Mat dst, int d,
                           double sigmaColor, double sigmaSpace,
                           int borderType)
```

В первом параметре указывается исходное изображение (глубина 8 битов или вещественные значения), содержащее один или три канала, а во втором — матрица, в которую будет записан результат операции. Параметр `d` задает диаметр окрестности пиксела (рекомендуемое значение: `5`). Если указано отрицательное число, то значение вычисляется из значения параметра `sigmaSpace`. Параметр `sigmaColor` определяет стандартное отклонение в цветовом пространстве, а параметр `sigmaSpace` — стандартное отклонение в пространстве координат. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`.

Пример двустороннего сглаживания по методу `bilateralFilter()` приведен в листинге 7.3.

Листинг 7.3. Метод `bilateralFilter()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.bilateralFilter(img, img2, 5, 5 * 2, 5 * 2);
CvUtilsFX.showImage(img2, "d = 5");
Mat img3 = new Mat();
Imgproc.bilateralFilter(img, img3, 9, 9 * 2, 9 * 2);
CvUtilsFX.showImage(img3, "d = 9");
img.release();
img2.release();
img3.release();
```

7.1.4. Метод *adaptiveBilateralFilter()*

Статический метод `adaptiveBilateralFilter()` из класса `Imgproc` сглаживает изображение, используя адаптивный двусторонний фильтр. Обратите внимание, метод доступен только в версии 2.4. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void adaptiveBilateralFilter(Mat src, Mat dst, Size ksize,
               double sigmaSpace)
public static void adaptiveBilateralFilter(Mat src, Mat dst, Size ksize,
               double sigmaSpace, double maxSigmaColor, Point anchor)
public static void adaptiveBilateralFilter(Mat src, Mat dst, Size ksize,
               double sigmaSpace, double maxSigmaColor, Point anchor,
               int borderType)
```

В первом параметре указывается исходное изображение, а во втором — матрица, в которую будет записан результат операции. Параметр `ksize` задает размеры ядра фильтра. Параметр `maxSigmaColor` определяет максимальное отклонение в цветовом пространстве (значение по умолчанию: `20`), а параметр `sigmaSpace` — стандартное отклонение в пространстве координат. В параметре `anchor` можно указать положение опорной точки. По умолчанию параметр `anchor` имеет значение `Point(-1, -1)`, которое означает, что опорная точка находится в центре ядра. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`.

Пример сглаживания по методу `adaptiveBilateralFilter()` приведен в листинге 7.4.

Листинг 7.4. Метод `adaptiveBilateralFilter()`

```
// Только в версии 2.4
Mat img = Highgui.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.adaptiveBilateralFilter(img, img2, new Size(5, 5), 5);
CvUtilsFX.showImage(img2, "Size(5, 5), 5");
Mat img3 = new Mat();
Imgproc.adaptiveBilateralFilter(img, img3, new Size(5, 5), 5, 50,
                                new Point(-1, -1));
CvUtilsFX.showImage(img3, "Size(5, 5), 5, 50");
img.release();
img2.release();
img3.release();
```

7.1.5. Медианный фильтр

Медианный фильтр реализуется с помощью статического метода `medianBlur()` из класса `Imgproc`. Обратите внимание: метод вычисляет не свертку, а медиану (выбирается средний элемент, и его значение становится результатом операции). Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static void medianBlur(Mat src, Mat dst, int ksize)
```

В первом параметре указывается исходное изображение, содержащее `1`, `3` или `4` канала (если значение параметра `ksize` равно `3` или `5`, то глубина может быть `CV_8U`, `CV_16U` или `CV_32F`, при значениях больше `5` — только `CV_8U`), а во втором — матрица, в которую будет записан результат операции. В параметре `ksize` указывается размер ядра фильтра (положительное нечетное значение больше `1`).

Пример фильтрации по методу `medianBlur()` приведен в листинге 7.5.

Листинг 7.5. Метод `medianBlur()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.medianBlur(img, img2, 3);
CvUtilsFX.showImage(img2, "3");
Mat img3 = new Mat();
```

```
Imgproc.medianBlur(img, img3, 5);
CvUtilsFX.showImage(img3, "5");
Mat img4 = new Mat();
Imgproc.medianBlur(img, img4, 45);
CvUtilsFX.showImage(img4, "45");
img.release();
img2.release();
img3.release();
img4.release();
```

7.1.6. Метод *boxFilter()*

Форматы метода `boxFilter()`:

```
import org.opencv.imgproc.Imgproc;
public static void boxFilter(Mat src, Mat dst, int ddepth, Size ksize)
public static void boxFilter(Mat src, Mat dst, int ddepth, Size ksize,
                      Point anchor, boolean normalize)
public static void boxFilter(Mat src, Mat dst, int ddepth, Size ksize,
                      Point anchor, boolean normalize, int borderType)
```

В первом параметре указывается исходное изображение, а во втором — матрица, в которую будет записан результат операции. Параметр `ddepth` задает глубину итоговой матрицы. Если указать значение `-1`, то будет использоваться глубина исходной матрицы. В параметре `ksize` указываются размеры ядра фильтра: чем больше размеры ядра, тем больше размытие. Параметр `anchor` задает положение опорной точки. По умолчанию используется значение `Point(-1, -1)`, которое означает, что опорная точка находится в центре ядра. Если в параметре `normalize` указано значение `true`, то результат будет точно такой же, как и при использовании метода `blur()`. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`.

Пример фильтрации по методу `boxFilter()` приведен в листинге 7.6.

Листинг 7.6. Метод `boxFilter()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat img2 = new Mat();
Imgproc.boxFilter(img, img2, -1, new Size(5, 5),
                  new Point(-1, -1), true);
CvUtilsFX.showImage(img2, "Size(5, 5)");
Mat img3 = new Mat();
Imgproc.boxFilter(img, img3, -1, new Size(45, 45),
                  new Point(-1, -1), true);
```

```
CvUtilsFX.showImage(img3, "Size(45, 45)");
img.release();
img2.release();
img3.release();
```

7.2. Методы *filter2D()* и *sepFilter2D()*

Статический метод `filter2D()` из класса `Imgproc` позволяет применить к изображению фильтр с произвольными значениями. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void filter2D(Mat src, Mat dst, int ddepth, Mat kernel)
public static void filter2D(Mat src, Mat dst, int ddepth, Mat kernel,
                    Point anchor, double delta)
public static void filter2D(Mat src, Mat dst, int ddepth, Mat kernel,
                    Point anchor, double delta, int borderType)
```

В первом параметре указывается исходное изображение, а во втором — матрица, в которую будет записан результат операции. Параметр `ddepth` задает глубину итоговой матрицы. Если указать значение `-1`, то будет использоваться глубина исходной матрицы. В параметре `kernel` указывается матрица с ядром свертки (ядром корреляции), содержащая один канал с вещественными значениями. Параметр `anchor` задает положение опорной точки. По умолчанию используется значение `Point(-1, -1)`, которое означает, что опорная точка находится в центре ядра. Значение, указанное в параметре `delta`, будет прибавлено к результату. По умолчанию используется значение `0`. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`.

Применить произвольный *сепарабельный фильтр* позволяет статический метод `sepFilter2D()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void sepFilter2D(Mat src, Mat dst, int ddepth,
                               Mat kernelX, Mat kernelY)
public static void sepFilter2D(Mat src, Mat dst, int ddepth,
                               Mat kernelX, Mat kernelY,
                               Point anchor, double delta)
public static void sepFilter2D(Mat src, Mat dst, int ddepth,
                               Mat kernelX, Mat kernelY,
                               Point anchor, double delta,
                               int borderType)
```

В параметре `kernelX` указывается одномерная матрица с ядром для фильтрации каждой строки, а в параметре `kernelY` — одномерная матрица с ядром для фильтрации каждого столбца. Остальные параметры соответствуют одноименным параметрам метода `filter2D()`.

Сепарабельный фильтр работает следующим образом: вначале каждая строка изображения фильтруется с помощью ядра `kernelX`, а затем каждый столбец результата

фильтруется с помощью ядра `kernelY`. К результату прибавляется значение параметра `delta`. Примером сепарабельного фильтра является фильтр Гаусса.

Пример использования методов `filter2D()` (имитация результата выполнения метода `blur()`) и `sepFilter2D()` (имитация результата метода `GaussianBlur()`) приведен в листинге 7.7.

Листинг 7.7. Методы `filter2D()` и `sepFilter2D()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat kernel = Mat.ones(new Size(3, 3), CvType.CV_32FC1);
Core.divide(kernel, new Scalar(kernel.rows() * kernel.cols()),
            kernel);
System.out.println(kernel.dump());
/*
[0.11111111, 0.11111111, 0.11111111;
0.11111111, 0.11111111, 0.11111111;
0.11111111, 0.11111111, 0.11111111]*/
Mat img2 = new Mat();
Imgproc.filter2D(img, img2, -1, kernel);
CvUtilsFX.showImage(img2, "filter2D()");
Mat img3 = new Mat();
Imgproc.blur(img, img3, new Size(3, 3), new Point(-1, -1));
CvUtilsFX.showImage(img3, "blur()");

Mat kernelGaussian = Imgproc.getGaussianKernel(3, 0, CvType.CV_64F);
System.out.println(kernelGaussian.dump());
/*
[0.25;
0.5;
0.25]*/
Mat img4 = new Mat();
Imgproc.sepFilter2D(img, img4, -1, kernelGaussian, kernelGaussian);
CvUtilsFX.showImage(img4, "sepFilter2D()");
Mat img5 = new Mat();
Imgproc.GaussianBlur(img, img5, new Size(3, 3), 0);
CvUtilsFX.showImage(img5, "GaussianBlur()");
img.release();              img2.release();
img3.release();             img4.release();
img5.release();
kernel.release();           kernelGaussian.release();
```

7.3. Методы *dilate()* и *erode()*

Статический метод `dilate()` из класса `Imgproc` расширяет светлые области и сужает темные. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void dilate(Mat src, Mat dst, Mat kernel)
public static void dilate(Mat src, Mat dst, Mat kernel, Point anchor,
                  int iterations)
public static void dilate(Mat src, Mat dst, Mat kernel, Point anchor,
                  int iterations, int borderType, Scalar borderValue)
```

Статический метод `erode()` из класса `Imgproc` выполняет обратную операцию: расширяет темные области и сужает светлые. Форматы метода:

```
public static void erode(Mat src, Mat dst, Mat kernel)
public static void erode(Mat src, Mat dst, Mat kernel, Point anchor,
                  int iterations)
public static void erode(Mat src, Mat dst, Mat kernel, Point anchor,
                  int iterations, int borderType, Scalar borderValue)
```

В первом параметре указывается исходное изображение (тип `CV_8U`, `CV_16U`, `CV_16S`, `CV_32F` или `CV_64F`), а во втором — матрица, в которую будет записан результат операции. Параметр `kernel` задает матрицу с ядром. Если в параметре `kernel` указано значение `Mat()`, то создается матрица размером `3x3` с ядром прямоугольной формы. Создать произвольную матрицу с ядром позволяет статический метод `getStructuringElement()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static Mat getStructuringElement(int shape, Size ksize)
public static Mat getStructuringElement(int shape, Size ksize,
                                        Point anchor)
```

В параметре `shape` можно указать следующие константы из класса `Imgproc`:

- `MORPH_RECT` — в форме прямоугольника. Формат:

  ```
  public static final int MORPH_RECT
  ```

- `MORPH_ELLIPSE` — в форме эллипса, вписанного в прямоугольник указанного размера. Формат:

  ```
  public static final int MORPH_ELLIPSE
  ```

- `MORPH_CROSS` — крестообразная форма. Формат:

  ```
  public static final int MORPH_CROSS
  ```

В параметре `ksize` указываются размеры матрицы, а в параметре `anchor` — положение опорной точки. По умолчанию параметр `anchor` имеет значение `Point(-1, -1)`, которое означает, что опорная точка находится в центре ядра. Выведем результат выполнения метода `getStructuringElement()`:

```
Mat m = Imgproc.getStructuringElement(Imgproc.MORPH_RECT,
                                      new Size(3, 3));
System.out.println(m.dump());
/*
[  1,   1,   1;
   1,   1,   1;
   1,   1,   1]*/
m = Imgproc.getStructuringElement(Imgproc.MORPH_ELLIPSE,
                                  new Size(5, 5));
System.out.println(m.dump());
/*
[  0,   0,   1,   0,   0;
   1,   1,   1,   1,   1;
   1,   1,   1,   1,   1;
   1,   1,   1,   1,   1;
   0,   0,   1,   0,   0]*/
m = Imgproc.getStructuringElement(Imgproc.MORPH_CROSS,
                                  new Size(3, 3));
System.out.println(m.dump());
/*
[  0,   1,   0;
   1,   1,   1;
   0,   1,   0]*/
m = Imgproc.getStructuringElement(Imgproc.MORPH_CROSS,
                                  new Size(5, 5), new Point(3, 3));
System.out.println(m.dump());
/*
[  0,   0,   0,   1,   0;
   0,   0,   0,   1,   0;
   0,   0,   0,   1,   0;
   1,   1,   1,   1,   1;
   0,   0,   0,   1,   0]*/
```

Параметр `anchor` в методах `dilate()` и `erode()` задает положение опорной точки. По умолчанию используется значение `Point(-1, -1)`, которое означает, что опорная точка находится в центре ядра. Параметр `iterations` задает количество итераций (повторений). По умолчанию используется значение `1`. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_CONSTANT`. В этом случае в параметре `borderValue` можно указать цвет.

Пример морфологических преобразований по методам `dilate()` и `erode()` приведен в листинге 7.8, а результат выполнения кода из листинга 7.8 показан на рис. 7.1.

Листинг 7.8. Методы `dilate()` и `erode()`

```
Mat img = new Mat(150, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.putText(img, "OpenCV", new Point(90, 50),
                Core.FONT_HERSHEY_SIMPLEX, 1, CvUtils.COLOR_BLACK, 3);
```

```
Imgproc.rectangle(img, new Point(50, 70), new Point(250, 120),
                  CvUtils.COLOR_BLACK, 3);
CvUtilsFX.showImage(img, "normal");
Mat kernel = Imgproc.getStructuringElement(Imgproc.MORPH_RECT,
                                           new Size(3, 3));
Mat img2 = new Mat();
Imgproc.dilate(img, img2, kernel);
CvUtilsFX.showImage(img2, "dilate");
Mat img3 = new Mat();
Imgproc.erode(img, img3, kernel);
CvUtilsFX.showImage(img3, "erode");
```

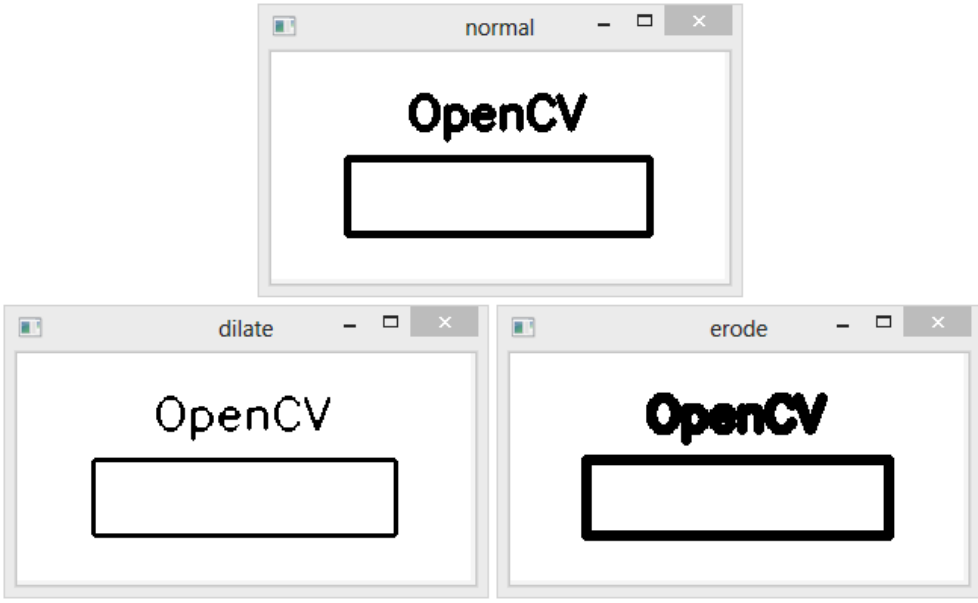


Рис. 7.1. Морфологические преобразования (листинг 7.8)

7.4. Метод *morphologyEx()*

Статический метод `morphologyEx()` из класса `Imgproc` позволяет выполнить морфологические преобразования. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void morphologyEx(Mat src, Mat dst, int op, Mat kernel)
public static void morphologyEx(Mat src, Mat dst, int op, Mat kernel,
                        Point anchor, int iterations)
public static void morphologyEx(Mat src, Mat dst, int op, Mat kernel,
                        Point anchor, int iterations, int borderType,
                        Scalar borderValue)
```

Рассмотренные в предыдущем разделе методы `dilate()` и `erode()` также выполняют морфологические преобразования. Эти методы активно используются методом `morphologyEx()`. Предназначения параметров метода `morphologyEx()` точно такие же, как и у одноименных параметров в методах `dilate()` и `erode()`, поэтому не будем повторяться, а рассмотрим только различия.

В параметре `op` можно указать следующие константы из класса `Imgproc`:

- `MORPH_OPEN` — выполняет следующую операцию:

  ```
  MORPH_OPEN = MORPH_DILATE( MORPH_ERODE(src) )
  ```

  Формат:

  ```
  public static final int MORPH_OPEN
  ```

- `MORPH_CLOSE` — выполняет следующую операцию:

  ```
  MORPH_CLOSE = MORPH_ERODE( MORPH_DILATE(src) )
  ```

  Формат:

  ```
  public static final int MORPH_CLOSE
  ```

- `MORPH_GRADIENT` — выполняет следующую операцию:

  ```
  MORPH_GRADIENT = MORPH_DILATE(src) - MORPH_ERODE(src)
  ```

  Формат:

  ```
  public static final int MORPH_GRADIENT
  ```

- `MORPH_TOPHAT` — выполняет следующую операцию:

  ```
  MORPH_TOPHAT = src - MORPH_OPEN(src)
  ```

  Формат:

  ```
  public static final int MORPH_TOPHAT
  ```

- `MORPH_BLACKHAT` — выполняет следующую операцию:

  ```
  MORPH_BLACKHAT = MORPH_CLOSE(src) - src
  ```

  Формат:

  ```
  public static final int MORPH_BLACKHAT
  ```

Пример морфологических преобразований по методу `morphologyEx()` приведен в листинге 7.9, а результат выполнения кода из листинга 7.9 показан на рис. 7.2.

Листинг 7.9. Метод `morphologyEx()`

```
Mat img = new Mat(150, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.putText(img, "OpenCV", new Point(90, 50),
                Core.FONT_HERSHEY_SIMPLEX, 1, CvUtils.COLOR_BLACK, 3);
Imgproc.rectangle(img, new Point(50, 70), new Point(250, 120),
                  CvUtils.COLOR_BLACK, 3);
```

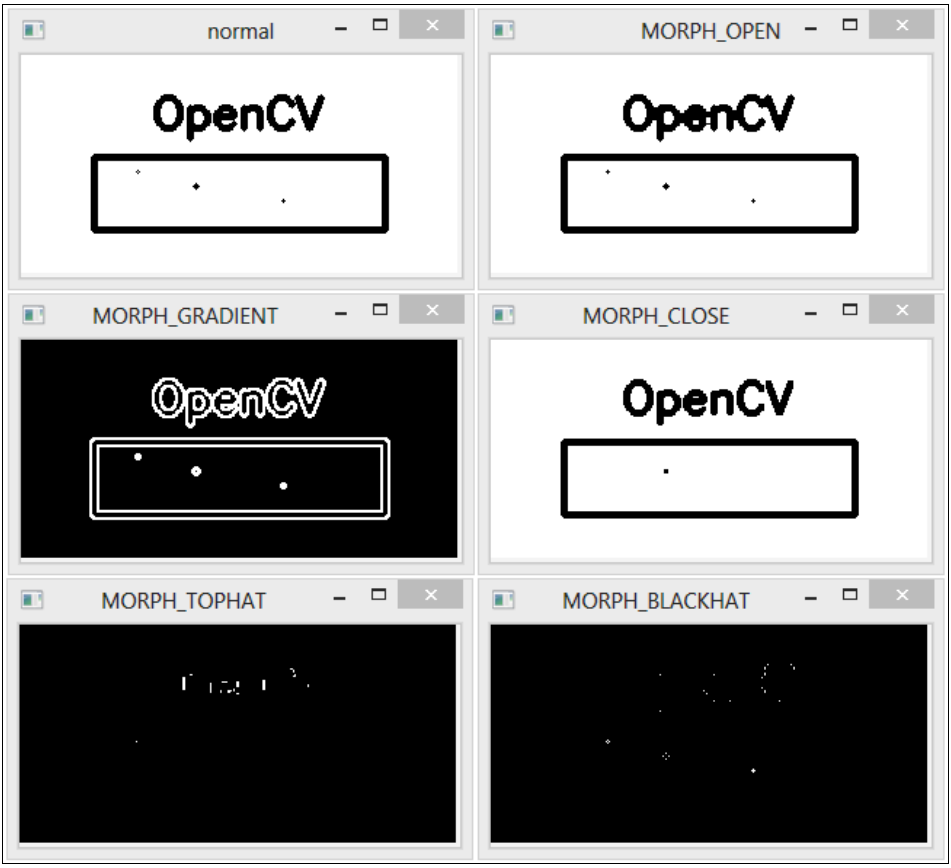


Рис. 7.2. Морфологические преобразования (листинг 7.9)

```
Imgproc.circle(img, new Point(80, 80), 1, CvUtils.COLOR_BLACK);
Imgproc.circle(img, new Point(120, 90), 2, CvUtils.COLOR_BLACK,
               Core.FILLED);
Imgproc.circle(img, new Point(180, 100), 1, CvUtils.COLOR_BLACK,
               Core.FILLED);
CvUtilsFX.showImage(img, "normal");
Mat kernel = Imgproc.getStructuringElement(Imgproc.MORPH_RECT,
                                           new Size(3, 3));
Mat img2 = new Mat();
Imgproc.morphologyEx(img, img2, Imgproc.MORPH_OPEN, kernel);
CvUtilsFX.showImage(img2, "MORPH_OPEN");
Mat img3 = new Mat();
Imgproc.morphologyEx(img, img3, Imgproc.MORPH_CLOSE, kernel);
CvUtilsFX.showImage(img3, "MORPH_CLOSE");
Mat img4 = new Mat();
Imgproc.morphologyEx(img, img4, Imgproc.MORPH_GRADIENT, kernel);
CvUtilsFX.showImage(img4, "MORPH_GRADIENT");
Mat img5 = new Mat();
```

```
Imgproc.morphologyEx(img, img5, Imgproc.MORPH_TOPHAT, kernel);
CvUtilsFX.showImage(img5, "MORPH_TOPHAT");
Mat img6 = new Mat();
Imgproc.morphologyEx(img, img6, Imgproc.MORPH_BLACKHAT, kernel);
CvUtilsFX.showImage(img6, "MORPH_BLACKHAT");
Mat img7 = new Mat();
Imgproc.dilate(img, img7, kernel);
CvUtilsFX.showImage(img7, "dilate");
Mat img8 = new Mat();
Imgproc.erode(img, img8, kernel);
CvUtilsFX.showImage(img8, "erode");
```

7.5. Гауссовы пирамиды

Статические методы `pyrDown()` и `pyrUp()` из класса `Imgproc` позволяют создать копии изображения уменьшенного и увеличенного размера соответственно. В процессе преобразования эти копии размываются. Если несколько раз применить метод к результирующему изображению, то получится целый набор изображений разных размеров, при наложении напоминающий пирамиду. Такие гауссовы пирамиды используются для поиска особых точек на изображении. Форматы методов `pyrDown()` и `pyrUp()`:

```
import org.opencv.imgproc.Imgproc;
public static void pyrDown(Mat src, Mat dst)
public static void pyrDown(Mat src, Mat dst, Size dstsize)
public static void pyrDown(Mat src, Mat dst, Size dstsize,
                           int borderType)

public static void pyrUp(Mat src, Mat dst)
public static void pyrUp(Mat src, Mat dst, Size dstsize)
public static void pyrUp(Mat src, Mat dst, Size dstsize, int borderType)
```

В первом параметре (`src`) указывается исходное изображение, а во втором (`dst`) — матрица, в которую будет записан результат операции. Параметр `dstsize` задает размеры результирующей матрицы, а параметр `borderType` — тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`. Тип `BORDER_CONSTANT` не поддерживается. В методе `pyrUp()` можно указать только тип `BORDER_DEFAULT`.

Метод `pyrDown()` размывает изображение, а затем уменьшает его размер следующим образом:

```
size = Size((src.cols + 1) / 2, (src.rows + 1) / 2)
```

Метод `pyrUp()` увеличивает размер следующим образом:

```
size = Size(src.cols * 2, src.rows * 2)
```

а затем размывает его.

Пример использования методов `pyrDown()` и `pyrUp()` приведен в листинге 7.10.

Листинг 7.10. Методы `pyrDown()` и `pyrUp()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto1.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
System.out.println(img.size());  // 3482x2321
Mat img2 = new Mat();
Imgproc.pyrDown(img, img2);
System.out.println(img2.size()); // 1741x1161
CvUtilsFX.showImage(img2, "- 1");
Mat img3 = new Mat();
Imgproc.pyrDown(img2, img3);
System.out.println(img3.size()); // 871x581
CvUtilsFX.showImage(img3, "- 2");
Mat img4 = new Mat();
Imgproc.pyrDown(img3, img4);
System.out.println(img4.size()); // 436x291
CvUtilsFX.showImage(img4, "- 3");
Mat img5 = new Mat();
Imgproc.pyrUp(img4, img5);
System.out.println(img5.size()); // 872x582
CvUtilsFX.showImage(img5, "- 3 + 1");
img.release();
img2.release();
img3.release();
img4.release();
img5.release();
```

7.6. Вычисление градиентов изображения

Градиент — это вектор, показывающий направление наибольшего возрастания некоторой величины. По его длине можно определить скорость роста этой величины в данном направлении. При работе с изображением в градациях серого мы имеем дело с матрицей чисел, описывающих яркость пиксела в определенной точке. В каждой точке изображения вектор градиента показывает направление наибольшего увеличения яркости, а его длина — величину изменения яркости. Для точки внутри области с постоянной яркостью вектор будет нулевым.

7.6.1. Методы *Sobel()* и *Scharr()*

Вычислить приближенное значение градиента яркости изображения позволяет статический метод `Sobel()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void Sobel(Mat src, Mat dst, int ddepth, int dx, int dy)
public static void Sobel(Mat src, Mat dst, int ddepth, int dx, int dy,
                  int ksize, double scale, double delta)
public static void Sobel(Mat src, Mat dst, int ddepth, int dx, int dy,
                  int ksize, double scale, double delta, int borderType)
```

В первом параметре указывается исходное изображение, а во втором — матрица, в которую будет записан результат операции. Параметр `ddepth` задает глубину итоговой матрицы. Если указано значение `-1`, то глубина будет соответствовать глубине исходного изображения. Следует учитывать, что вычисленное значение может быть как положительным, так и отрицательным. Если мы используем исходное изображение с глубиной `CV_8U`, то в параметре `ddepth` лучше явно указать глубину `CV_16S`, `CV_32F` или `CV_64F`, иначе отрицательное значение в результате нормализации станет равно `0`. Выполнить преобразование матрицы в тип `CV_8U` без потери отрицательных значений можно с помощью статического метода `convertScaleAbs()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void convertScaleAbs(Mat src, Mat dst)
public static void convertScaleAbs(Mat src, Mat dst, double alpha,
                                   double beta)
```

Параметр `dx` задает порядок производной по оси `X` (`0`, `1` или `2`), а параметр `dy` — по оси `Y`. Оба параметра не могут одновременно иметь значение `0`.

Параметр `ksize` задает размер ядра. Можно указать значения `1` (используется ядро `3x1` или `1x3`), `3`, `5` и `7`. Максимальный размер ядра хранится в константе `CV_MAX_SOBEL_KSIZE`. Формат:

```
public static final int CV_MAX_SOBEL_KSIZE
```

Выведем значение константы:

```
System.out.println(Imgproc.CV_MAX_SOBEL_KSIZE); // 7
```

Ядро Собела размером `3x3` для `X` (`dx = 1`, `dy = 0`) выглядит следующим образом:

```
-1   0   1
-2   0   2
-1   0   1
```

Для `Y` (`dx = 0`, `dy = 1`) так:

```
-1  -2  -1
 0   0   0
 1   2   1
```

В качестве значения параметра `ksize` можно указать константу `CV_SCHARR`, которая указывает, что должно использоваться ядро Щарра. Формат:

```
public static final int CV_SCHARR
```

Выведем значение константы:

```
System.out.println(Imgproc.CV_SCHARR); // -1
```

Ядро Щарра для X выглядит следующим образом (для Y получается транспонированием):

```
-3   0   3
-10  0  10
-3   0   3
```

Применить ядро Щарра позволяет также статический метод Scharr() из класса Imgproc. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void Scharr(Mat src, Mat dst, int ddepth, int dx, int dy)
public static void Scharr(Mat src, Mat dst, int ddepth, int dx, int dy,
                  double scale, double delta)
public static void Scharr(Mat src, Mat dst, int ddepth, int dx, int dy,
                  double scale, double delta, int borderType)
```

Получить значения для ядер Собела и Щарра позволяет статический метод getDerivKernels() из класса Imgproc. Форматы метода:

```
public static void getDerivKernels(Mat kx, Mat ky, int dx, int dy,
                           int ksize)
public static void getDerivKernels(Mat kx, Mat ky, int dx, int dy,
                           int ksize, boolean normalize, int ktype)
```

В параметре scale метода Sobel() можно дополнительно указать масштабный коэффициент (значение по умолчанию: 1), а в параметре delta — значение, которое будет прибавлено к результату операции (значение по умолчанию: 0). В параметре borderType можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение BORDER_DEFAULT.

В версии 3.3 доступен статический метод spatialGradient(), который позволяет вычислить производную первого порядка сразу и по оси X, и по оси Y. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void spatialGradient(Mat src, Mat dx, Mat dy)
public static void spatialGradient(Mat src, Mat dx, Mat dy, int ksize)
public static void spatialGradient(Mat src, Mat dx, Mat dy, int ksize,
                                   int borderType)
```

Пример:

```
Imgproc.spatialGradient(imgGray, dstSobelX, dstSobelY);
```

Этот код заменяет следующую последовательность вызовов метода Sobel():

```
Imgproc.Sobel(imgGray, dstSobelX, CvType.CV_16S, 1, 0);
Imgproc.Sobel(imgGray, dstSobelY, CvType.CV_16S, 0, 1);
```

Чтобы вычислить градиент, нужно вначале рассчитать приближенные значения производных отдельно по оси X и по оси Y:

```
Mat dstSobelX = new Mat();
Imgproc.Sobel(imgGray, dstSobelX, CvType.CV_32F, 1, 0);
Mat dstSobelY = new Mat();
Imgproc.Sobel(imgGray, dstSobelY, CvType.CV_32F, 0, 1);
```

Далее для каждого элемента матрицы вычисление градиента производится следующим образом:

```
double[] dx = dstSobelX.get(i, j);
double[] dy = dstSobelY.get(i, j);
double grad = Math.sqrt(Math.pow(dx[0], 2) + Math.pow(dy[0], 2));
```

Вычислить значение градиента для всех элементов матрицы позволяет статический метод `magnitude()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void magnitude(Mat x, Mat y, Mat magnitude)
```

Пример:

```
Mat magnitude = new Mat();
Core.magnitude(dstSobelX, dstSobelY, magnitude);
```

Вычислить направление градиента можно так:

```
double angle = Math.toDegrees(Math.atan(dy[0] / dx[0]));
```

Вычислить направление градиента для всех элементов матрицы позволяет статический метод `phase()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void phase(Mat x, Mat y, Mat angle)
public static void phase(Mat x, Mat y, Mat angle,
                         boolean angleInDegrees)
```

Если в параметре `angleInDegrees` указать значение `true`, то угол будет возвращаться в градусах, а если `false` (значение по умолчанию) — то в радианах.

Пример:

```
Mat angle = new Mat();
Core.phase(dstSobelX, dstSobelY, angle, true);
```

Получить значение градиента и направление для всех элементов матрицы позволяет статический метод `cartToPolar()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void cartToPolar(Mat x, Mat y, Mat magnitude, Mat angle)
public static void cartToPolar(Mat x, Mat y, Mat magnitude, Mat angle,
                               boolean angleInDegrees)
```

Если в параметре `angleInDegrees` указать значение `true`, то угол будет возвращаться в градусах, а если `false` (значение по умолчанию) — то в радианах.

Пример:

```
Mat magnitude = new Mat();
Mat angle = new Mat();
Core.cartToPolar(dstSobelX, dstSobelY, magnitude, angle, true);
```

Выполнить обратную операцию позволяет статический метод `polarToCart()` из класса `Core`. Форматы метода:

```
import org.opencv.core.Core;
public static void polarToCart(Mat magnitude, Mat angle, Mat x, Mat y)
public static void polarToCart(Mat magnitude, Mat angle, Mat x, Mat y,
                               boolean angleInDegrees)
```

Пример использования методов `Sobel()` и `Scharr()` для поиска границ объектов приведен в листинге 7.11, а результат выполнения кода из листинга 7.11 показан на рис. 7.3.

Листинг 7.11. Методы `Sobel()` и `Scharr()`

```
Mat img = new Mat(150, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.rectangle(img, new Point(20, 20), new Point(120, 70),
                  CvUtils.COLOR_GREEN, Core.FILLED);
Imgproc.line(img, new Point(20, 100), new Point(120, 100),
             CvUtils.COLOR_RED, 1);
Imgproc.line(img, new Point(20, 120), new Point(120, 120),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(150, 20), new Point(150, 100),
             CvUtils.COLOR_RED, 1);
Imgproc.line(img, new Point(170, 20), new Point(170, 100),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(200, 20), new Point(280, 100),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(280, 20), new Point(200, 100),
             CvUtils.COLOR_RED, 3);
CvUtilsFX.showImage(img, "Оригинал");

Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);

Mat dstSobelX = new Mat();
Imgproc.Sobel(imgGray, dstSobelX, CvType.CV_32F, 1, 0);
Mat dstSobelY = new Mat();
Imgproc.Sobel(imgGray, dstSobelY, CvType.CV_32F, 0, 1);

Mat imgSobelX = new Mat();
Core.convertScaleAbs(dstSobelX, imgSobelX);
Mat imgSobelY = new Mat();
Core.convertScaleAbs(dstSobelY, imgSobelY);
CvUtilsFX.showImage(imgSobelX, "Sobel X");
CvUtilsFX.showImage(imgSobelY, "Sobel Y");

Mat dstScharrX = new Mat();
Imgproc.Scharr(imgGray, dstScharrX, CvType.CV_32F, 1, 0);
```

```
Mat dstScharrY = new Mat();
Imgproc.Scharr(imgGray, dstScharrY, CvType.CV_32F, 0, 1);

Mat imgScharrX = new Mat();
Core.convertScaleAbs(dstScharrX, imgScharrX);
Mat imgScharrY = new Mat();
Core.convertScaleAbs(dstScharrY, imgScharrY);
CvUtilsFX.showImage(imgScharrX, "Scharr X");
CvUtilsFX.showImage(imgScharrY, "Scharr Y");

img.release();         imgGray.release();
dstSobelX.release();   dstSobelY.release();
imgSobelX.release();   imgSobelY.release();
dstScharrX.release(); dstScharrY.release();
imgScharrX.release(); imgScharrY.release();
```

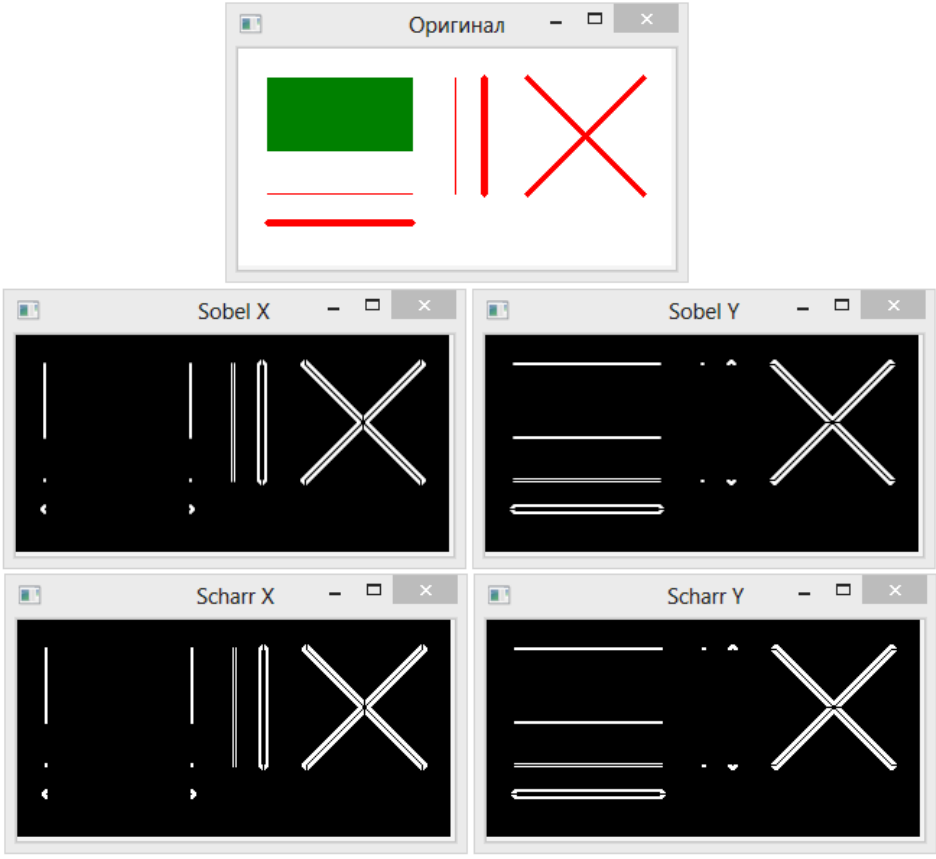


Рис. 7.3. Результат выполнения кода из листинга 7.11

7.6.2. Метод *Laplacian()*

Статический метод `Laplacian()` из класса `Imgproc` вычисляет Лапласиан изображения. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void Laplacian(Mat src, Mat dst, int ddepth)
public static void Laplacian(Mat src, Mat dst, int ddepth, int ksize,
                      double scale, double delta)
public static void Laplacian(Mat src, Mat dst, int ddepth, int ksize,
                      double scale, double delta, int borderType)
```

В первом параметре указывается исходное изображение, а во втором — матрица, в которую будет записан результат операции. Параметр `ddepth` задает глубину итоговой матрицы. Параметр `ksize` задает размер ядра фильтра (положительное нечетное число). Если указано значение `1` или параметр не указан, то ядро будет выглядеть следующим образом:

```
0   1   0
1  -4   1
0   1   0
```

В параметре `scale` можно дополнительно указать масштабный коэффициент (значение по умолчанию: `1`), а в параметре `delta` — значение, которое будет прибавлено к результату операции (значение по умолчанию: `0`). В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`.

Пример поиска границ объектов по методу `Laplacian()` приведен в листинге 7.12, а результат выполнения кода листинга 7.12 показан на рис. 7.4.

Листинг 7.12. Метод `Laplacian()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");

Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);

Mat dstLaplacian = new Mat();
Imgproc.Laplacian(imgGray, dstLaplacian, CvType.CV_32F);
Mat imgLaplacian = new Mat();
Core.convertScaleAbs(dstLaplacian, imgLaplacian);
CvUtilsFX.showImage(imgLaplacian, "Laplacian ksize=1");
```

```
Mat dstLaplacian2 = new Mat();
Imgproc.Laplacian(imgGray, dstLaplacian2, CvType.CV_32F, 3, 1, 0);
Mat imgLaplacian2 = new Mat();
Core.convertScaleAbs(dstLaplacian2, imgLaplacian2);
CvUtilsFX.showImage(imgLaplacian2, "Laplacian ksize=3");

img.release();              imgGray.release();
dstLaplacian.release();     dstLaplacian.release();
imgLaplacian2.release();    imgLaplacian2.release();
```

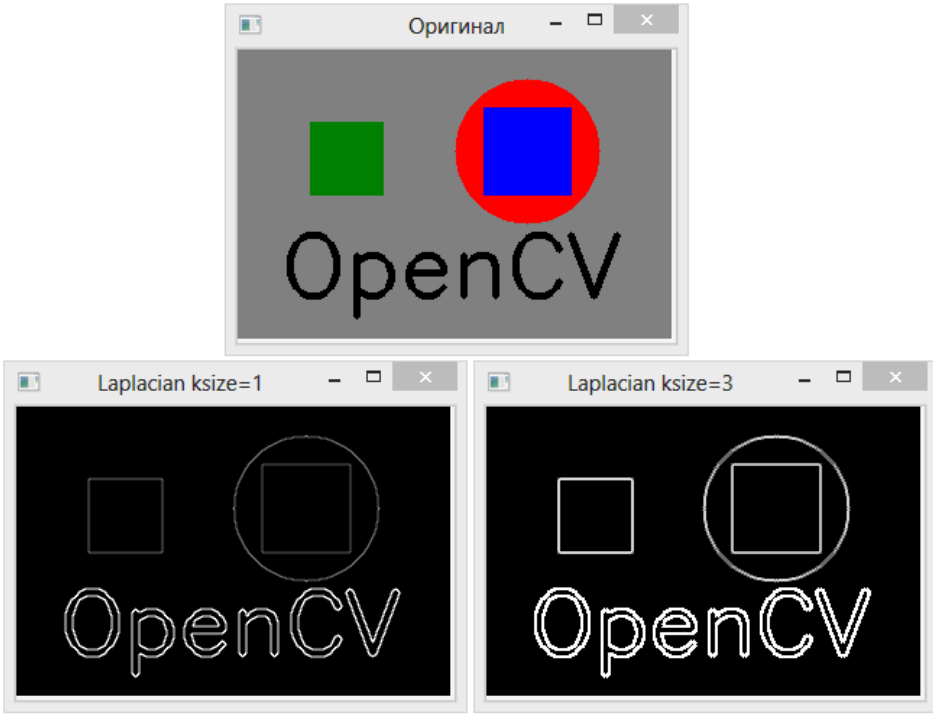


Рис. 7.4. Результат выполнения кода из листинга 7.12

7.7. Фильтр Габора

Создать ядро фильтра Габора позволяет статический метод `getGaborKernel()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static Mat getGaborKernel(Size ksize, double sigma, double theta,
             double lambd, double gamma)
public static Mat getGaborKernel(Size ksize, double sigma, double theta,
             double lambd, double gamma, double psi, int ktype)
```

Параметр `ksize` задает размеры ядра фильтра (размеры должны быть нечетными), параметр `sigma` — стандартное отклонение Гаусса, параметр `theta` — угол поворота ядра в радианах, параметр `lambd` — длину волны, параметр `gamma` — пространственное соотношение сторон, параметр `psi` — фазовый сдвиг в радианах (значение по умолчанию: `PI * 0.5`), а параметр `ktype` — глубину итоговой матрицы (`CV_32F` или `CV_64F`, значение по умолчанию: `CV_64F`).

Применить фильтр Габора к изображению позволяет метод `filter2D()` (см. *разд. 7.2*). Обычно создают несколько фильтров, повернутых на разный угол (параметр `theta`), применяют их к изображению, а затем складывают результат в единое изображение (листинг 7.13).

Листинг 7.13. Фильтр Габора

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
Mat imgF = new Mat();
img.convertTo(imgF, CvType.CV_32F, 1.0 / 255);
CvUtilsFX.showImage(imgF, "Оригинал");

int ksize = 31;
double sigma = 1.5;
double lambd = 0.3;
double gamma = 0.1;
Mat kernel_0 = Imgproc.getGaborKernel(new Size(ksize, ksize),
                      sigma, Math.toRadians(0), lambd, gamma);
Mat kernel_45 = Imgproc.getGaborKernel(new Size(ksize, ksize),
                      sigma, Math.toRadians(45), lambd, gamma);
Mat kernel_90 = Imgproc.getGaborKernel(new Size(ksize, ksize),
                      sigma, Math.toRadians(90), lambd, gamma);
Mat kernel_135 = Imgproc.getGaborKernel(new Size(ksize, ksize),
                      sigma, Math.toRadians(135), lambd, gamma);
// Просмотр ядра
Mat img_45 = new Mat();
Core.normalize(kernel_45, img_45, 0, 1, Core.NORM_MINMAX,
              CvType.CV_32F);
CvUtilsFX.showImage(img_45, "kernel_45");
// Применяем фильтры
Mat result_0 = new Mat();
Imgproc.filter2D(imgF, result_0, CvType.CV_32F, kernel_0);
CvUtilsFX.showImage(result_0, "kernel_0");
Mat result_45 = new Mat();
Imgproc.filter2D(imgF, result_45, CvType.CV_32F, kernel_45);
CvUtilsFX.showImage(result_45, "kernel_45");
```

```
Mat result_90 = new Mat();
Imgproc.filter2D(imgF, result_90, CvType.CV_32F, kernel_90);
CvUtilsFX.showImage(result_90, "kernel_90");
Mat result_135 = new Mat();
Imgproc.filter2D(imgF, result_135, CvType.CV_32F, kernel_135);
CvUtilsFX.showImage(result_135, "kernel_135");
// Создаем итоговое изображение
Mat result = new Mat(imgF.size(), CvType.CV_32FC1, new Scalar(0));
Core.addWeighted(result, 1, result_0, 1, 0, result);
Core.addWeighted(result, 1, result_45, 1, 0, result);
Core.addWeighted(result, 1, result_90, 1, 0, result);
Core.addWeighted(result, 1, result_135, 1, 0, result);
CvUtilsFX.showImage(result, "Результат");

img.release(); imgF.release(); img_45.release();
kernel_0.release();   kernel_45.release(); kernel_90.release();
kernel_135.release();
result_0.release();   result_45.release(); result_90.release();
result_135.release(); result.release();
```

7.8. Повышение резкости

Стандартного метода, позволяющего повысить резкость изображения, в библиотеке OpenCV нет. Попробуем создать его самостоятельно. Первый способ повышения резкости состоит в применении фильтра, имеющего следующее ядро:

```
  0.1111, -0.8889,  0.1111
 -0.8889,  4.1111, -0.8889
  0.1111, -0.8889,  0.1111
```

Универсальным способ назвать нельзя. Изменим ядро и введем коэффициент:

```
double k = 0.5;
Mat kernel2 = new Mat(3, 3, CvType.CV_32FC1);
kernel2.put(0, 0,
    -k,      k - 1,  -k,
     k - 1,  k + 5,   k - 1,
    -k,      k - 1,  -k);
Core.divide(kernel2, new Scalar(k + 1), kernel2);
```

Или так:

```
double n = 1.0;
Mat kernel3 = new Mat(3, 3, CvType.CV_32FC1);
kernel3.put(0, 0,
   n - 1,  -n,       n - 1,
  -n,       n + 5,  -n,
   n - 1,  -n,       n - 1);
Core.divide(kernel3, new Scalar(n + 1), kernel3);
```

При увеличении резкости этими способами вокруг границ объектов могут возникать артефакты в виде белых линий. Более качественный результат дает метод, называемый «Unsharp Mask». В этом случае создается копия исходного изображения, размытая фильтром Гаусса. Эта копия прибавляется к оригиналу следующим образом:

```
Mat img2 = new Mat();
Imgproc.GaussianBlur(img, img2, new Size(9, 9), 0);
Mat result4 = new Mat();
Core.addWeighted(img, 1.5, img2, -0.5, 0, result4);
```

В результате повышается резкость на границах объектов без появления артефактов, но изображение при этом может темнеть.

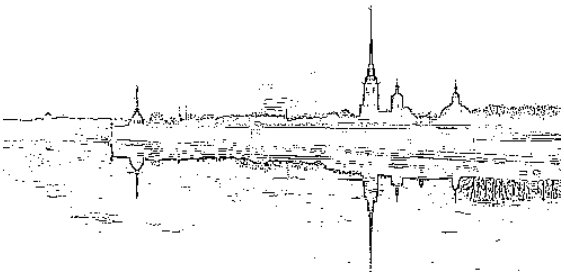
Полный код повышения резкости изображения способами, рассмотренными в этом разделе, приведен в листинге 7.14.

Листинг 7.14. Повышение резкости

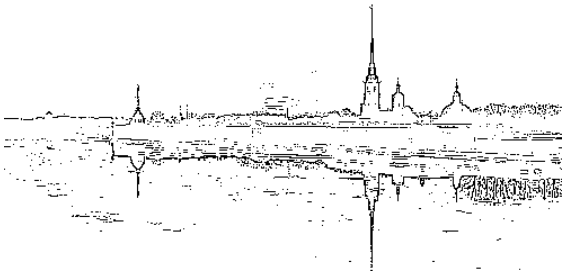
```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");
// Способ 1
Mat kernel = new Mat(3, 3, CvType.CV_32FC1);
kernel.put(0, 0,
    0.1111, -0.8889,  0.1111,
   -0.8889,  4.1111, -0.8889,
    0.1111, -0.8889,  0.1111);
Mat result = new Mat();
Imgproc.filter2D(img, result, -1, kernel);
CvUtilsFX.showImage(result, "Результат kernel1");
// Способ 2
double k = 0.5;
Mat kernel2 = new Mat(3, 3, CvType.CV_32FC1);
kernel2.put(0, 0,
    -k,      k - 1,  -k,
     k - 1,  k + 5,   k - 1,
    -k,      k - 1,  -k);
Core.divide(kernel2, new Scalar(k + 1), kernel2);
System.out.println(kernel2.dump());
Mat result2 = new Mat();
Imgproc.filter2D(img, result2, -1, kernel2);
CvUtilsFX.showImage(result2, "Результат kernel2");
// Способ 3
double n = 1.0;
```

```
Mat kernel3 = new Mat(3, 3, CvType.CV_32FC1);
kernel3.put(0, 0,
   n - 1,  -n,       n - 1,
  -n,       n + 5,  -n,
   n - 1,  -n,       n - 1);
Core.divide(kernel3, new Scalar(n + 1), kernel3);
Mat result3 = new Mat();
Imgproc.filter2D(img, result3, -1, kernel3);
CvUtilsFX.showImage(result3, "Результат kernel3");
// Способ 4
Mat img2 = new Mat();
Imgproc.GaussianBlur(img, img2, new Size(9, 9), 0);
Mat result4 = new Mat();
Core.addWeighted(img, 1.5, img2, -0.5, 0, result4);
CvUtilsFX.showImage(result4, "Unsharp mask");

img.release();  result.release();  result2.release();
img2.release(); result3.release(); result4.release();
```

ЧАСТЬ III



Компьютерное зрение



ГЛАВА 8

Поиск объектов

В предыдущих главах мы научились обрабатывать изображения. Начиная с этой главы, мы приступаем к анализу изображений и нахождению характерных различий одного изображения от другого, а также к поиску объектов.

8.1. Поиск контуров

Любой объект на изображении имеет границы, которые мы воспринимаем как резкий перепад яркости между двумя областями. Причем границы могут описывать как внешние контуры объекта, так и, например, текстуру внутри объекта.

8.1.1. Метод *Canny()*: выделение границ

Выделить границы объектов на изображении позволяет статический метод `Canny()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void Canny(Mat image, Mat edges, double threshold1,
                         double threshold2)
public static void Canny(Mat image, Mat edges, double threshold1,
                         double threshold2, int apertureSize,
                         boolean L2gradient)
```

В версии 3.3 доступны также следующие форматы метода `Canny()`:

```
public static void Canny(Mat dx, Mat dy, Mat edges, double threshold1,
                         double threshold2)
public static void Canny(Mat dx, Mat dy, Mat edges, double threshold1,
                         double threshold2, boolean L2gradient)
```

В параметре `image` указывается исходное изображение в оттенках серого (8 битов, один канал). В матрицу, указанную в параметре `edges`, будет записан результат операции в виде черно-белого изображения. Параметр `threshold1` задает минимальное пороговое значение, а параметр `threshold2` — максимальное пороговое

значение. В параметре `apertureSize` можно дополнительно указать размер ядра фильтра для метода `Sobel()` (значение по умолчанию: `3`). Если в параметре `L2gradient` указать значение `true`, то вычисления величины градиента будут более точными (значение по умолчанию: `false`).

К сожалению, не существует универсального набора пороговых значений, а от них зависит качество выделения границ. Для каждого изображения придется подбирать пороговые значения вручную. Если нужно что-то универсальное, то можно предварительно преобразовать изображение в оттенках серого в черно-белое, прибегнув к методу `threshold()` в сочетании с флагом `THRESH_OTSU` или к методу `adaptiveThreshold()` (листинг 8.1). Результат использования метода `Canny()` показан на рис. 8.1.

Листинг 8.1. Выделение границ

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(imgGray, "GRAY");
Mat edges = new Mat();
Imgproc.Canny(imgGray, edges, 80, 200);
CvUtilsFX.showImage(edges, "Canny");
Mat img3 = new Mat();
Imgproc.threshold(imgGray, img3, 100, 255,
                  Imgproc.THRESH_BINARY | Imgproc.THRESH_OTSU);
Mat edges2 = new Mat();
Imgproc.Canny(img3, edges2, 80, 200);
CvUtilsFX.showImage(edges2, "Canny + THRESH_OTSU");
Mat img4 = new Mat();
Imgproc.adaptiveThreshold(imgGray, img4, 255,
                          Imgproc.ADAPTIVE_THRESH_MEAN_C,
                          Imgproc.THRESH_BINARY, 3, 5);
Mat edges3 = new Mat();
Imgproc.Canny(img4, edges3, 80, 200);
CvUtilsFX.showImage(edges3, "Canny + adaptiveThreshold");
img.release(); img3.release(); img4.release();
imgGray.release();
edges.release(); edges2.release(); edges3.release();
```

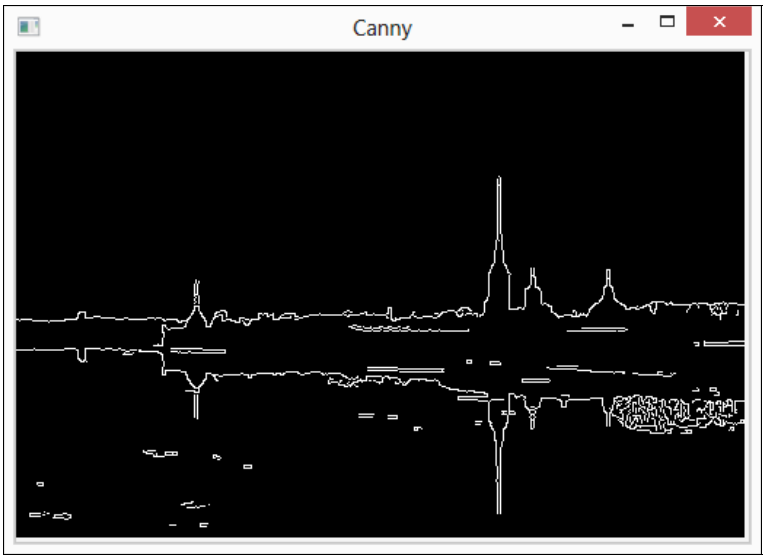


Рис. 8.1. Выделение границ (листинг 8.1)

8.1.2. Метод *findContours()*: поиск контуров

Выделив границы, можно выполнить поиск контуров. Для этого предназначен статический метод `findContours()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void findContours(Mat image, List<MatOfPoint> contours,
                                Mat hierarchy, int mode, int method)
public static void findContours(Mat image, List<MatOfPoint> contours,
                                Mat hierarchy, int mode, int method,
                                Point offset)
```

В первом параметре указывается исходное черно-белое изображение (8 битов, один канал). Любое значение больше `1`, считается `1`, а нулевые значения так и остаются нулевыми. Если в параметре `mode` указана константа `RETR_CCOMP`, то исходная матрица может иметь тип `CV_32SC1`. Обратите внимание: метод `findContours()` в версии 2.4 изменяет исходную матрицу — если это нежелательно, то следует передать копию матрицы.

Во втором параметре указывается ссылка на список, в который будут добавляться найденные контуры. Параметр `hierarchy` задает ссылку на матрицу, в которую будет записана информация об уровне вложенности контура. Эта матрица будет иметь столько же элементов, что и список с найденными контурами. Если контур не является вложенным, то значение будет отрицательным.

Параметр `mode` задает режим поиска контуров. Можно указать следующие константы из класса `Imgproc`:

- `RETR_EXTERNAL` — найти только крайние внешние контуры. Формат:

  ```
  public static final int RETR_EXTERNAL
  ```

- `RETR_LIST` — найти все контуры без установления иерархии. Формат:

  ```
  public static final int RETR_LIST
  ```

- `RETR_CCOMP` — найти все контуры и организовать их в двухуровневую структуру. Формат:

  ```
  public static final int RETR_CCOMP
  ```

- `RETR_TREE` — найти все контуры и организовать полную иерархию вложенных контуров. Формат:

  ```
  public static final int RETR_TREE
  ```

Выведем значения констант:

```
System.out.println(Imgproc.RETR_EXTERNAL); // 0
System.out.println(Imgproc.RETR_LIST);     // 1
System.out.println(Imgproc.RETR_CCOMP);    // 2
System.out.println(Imgproc.RETR_TREE);     // 3
```

Параметр `method` задает способ описания найденных контуров. Можно указать следующие константы из класса `Imgproc`:

- `CHAIN_APPROX_NONE` — сохраняются все точки контура. Формат:

  ```
  public static final int CHAIN_APPROX_NONE
  ```

- `CHAIN_APPROX_SIMPLE` — горизонтальные, вертикальные и диагональные сегменты сжимаются, и указываются только их конечные точки. Например, прямая линия будет закодирована двумя точками. Формат:

  ```
  public static final int CHAIN_APPROX_SIMPLE
  ```

- `CHAIN_APPROX_TC89_KCOS` и `CHAIN_APPROX_TC89_L1` — используется алгоритм Teh-Chin. Форматы:

  ```
  public static final int CHAIN_APPROX_TC89_KCOS
  public static final int CHAIN_APPROX_TC89_L1
  ```

Выведем значения констант:

```
System.out.println(Imgproc.CHAIN_APPROX_NONE);      // 0
System.out.println(Imgproc.CHAIN_APPROX_SIMPLE);    // 1
System.out.println(Imgproc.CHAIN_APPROX_TC89_KCOS); // 4
System.out.println(Imgproc.CHAIN_APPROX_TC89_L1);   // 3
```

Параметр `offset` задает смещение, на которое будут сдвинуты точки контура. По умолчанию используется значение `Point(0, 0)`.

8.1.3. Метод *drawContours()*: отрисовка контура

После нахождения контуров можно выполнять с ними различные операции. Первое, что хочется сделать, — это увидеть, какие контуры были найдены. Выполнить отрисовку найденных контуров позволяет статический метод `drawContours()` из класса `Imgproc`.

Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void drawContours(Mat image, List<MatOfPoint> contours,
                        int contourIdx, Scalar color)
public static void drawContours(Mat image, List<MatOfPoint> contours,
                        int contourIdx, Scalar color, int thickness)
public static void drawContours(Mat image, List<MatOfPoint> contours,
                        int contourIdx, Scalar color, int thickness,
                        int lineType, Mat hierarchy, int maxLevel,
                        Point offset)
```

В первом параметре указывается матрица, внутри которой будут рисоваться контуры. Во втором параметре указывается ссылка на список с найденными контурами, а в третьем — индекс рисуемого контура из этого списка (если указано отрицательное число, то рисуются все контуры). Параметр `color` задает цвет линии, а параметр `thickness` — толщину линии (если указано значение `Core.FILLED`, то выполняется заливка содержимого контуров). В параметре `lineType` указывается тип линии: константы `LINE_4`, `LINE_8` (значение по умолчанию) или `LINE_AA` (со сглаживанием). В параметре `hierarchy` можно указать ссылку на матрицу с иерархией контуров, а в параметре `maxLevel` — максимальный уровень иерархии. Параметр `offset` задает смещение, на которое будут сдвинуты точки контура. По умолчанию используется значение `Point(0, 0)`.

Пример поиска и отрисовки контуров приведен в листинге 8.2, а результат выполнения листинга 8.2 показан на рис. 8.2.

Листинг 8.2. Поиск и отрисовка контуров

```
Mat img = new Mat(200, 300, CvType.CV_8UC3, CvUtils.COLOR_GRAY);
Imgproc.rectangle(img, new Point(50, 50), new Point(100, 100),
                  CvUtils.COLOR_GREEN, Core.FILLED);
Imgproc.circle(img, new Point(200, 70), 50, CvUtils.COLOR_RED,
               Core.FILLED);
Imgproc.rectangle(img, new Point(170, 40), new Point(230, 100),
                  CvUtils.COLOR_BLUE, Core.FILLED);
Imgproc.putText(img, "OpenCV", new Point(30, 170),
             Core.FONT_HERSHEY_SIMPLEX, 2, CvUtils.COLOR_BLACK, 3);
CvUtilsFX.showImage(img, "Оригинал");

Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
Mat edges = new Mat();
Imgproc.Canny(imgGray, edges, 80, 200);
CvUtilsFX.showImage(edges, "Canny");
Mat edgesCopy = edges.clone(); // Создаем копию
ArrayList<MatOfPoint> contours = new ArrayList<MatOfPoint>();
Mat hierarchy = new Mat();
```

```
Imgproc.findContours(edgesCopy, contours, hierarchy,
                     Imgproc.RETR_TREE,
                     Imgproc.CHAIN_APPROX_SIMPLE);
System.out.println(contours.size());
System.out.println(hierarchy.size());
System.out.println(hierarchy.dump());
Imgproc.drawContours(img, contours, -1, CvUtils.COLOR_WHITE);
CvUtilsFX.showImage(img, "drawContours");
img.release();   imgGray.release();
edges.release(); edgesCopy.release();
hierarchy.release();
```

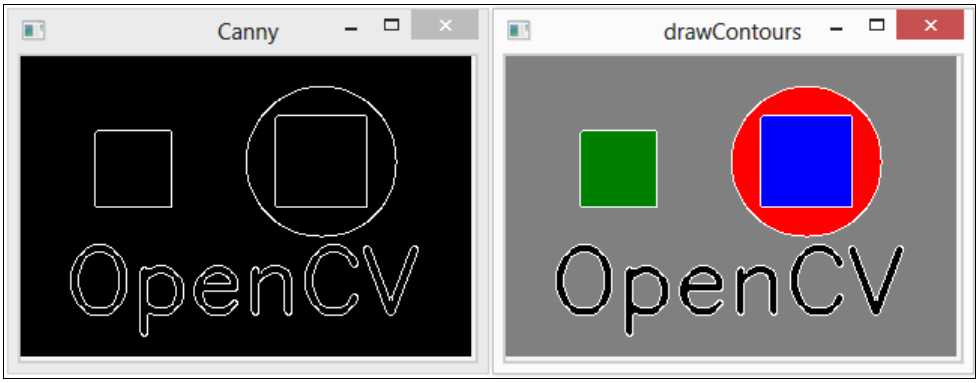


Рис. 8.2. Результат выполнения кода из листинга 8.2

8.1.4. Основные методы для работы с контурами

Для работы с контурами можно воспользоваться следующими методами из класса `Imgproc`:

- `boundingRect()` — возвращает прямоугольную область, в которую вписан контур. Формат метода:

  ```
  public static Rect boundingRect(MatOfPoint points)
  ```

- `minAreaRect()` — возвращает повернутую на некоторый угол прямоугольную область, в которую вписан контур. Формат метода:

  ```
  public static RotatedRect minAreaRect(MatOfPoint2f points)
  ```

- `minEnclosingCircle()` — возвращает координаты центра (параметр `center`) и радиус круга (параметр `radius`), в который вписан контур (параметр `points`). Формат метода:

  ```
  public static void minEnclosingCircle(MatOfPoint2f points,
                                        Point center, float[] radius)
  ```

Пример нахождения и отрисовки круга для контура с индексом `i`:

```
Point center = new Point();
float[] radius = new float[1];
Imgproc.minEnclosingCircle(
               new MatOfPoint2f(contours.get(i).toArray()),
               center, radius);
Imgproc.circle(img, center, (int) radius[0],
               CvUtils.COLOR_WHITE);
```

- `contourArea()` — возвращает площадь контура. Если в параметре `oriented` указано значение `true`, то возвращаемое значение может быть отрицательным. С помощью знака можно определить ориентацию контура. По умолчанию параметр `oriented` имеет значение `false`. Обратите внимание: в некоторых случаях метод выдает неправильный результат. Форматы метода:

```
public static double contourArea(Mat contour)
public static double contourArea(Mat contour, boolean oriented)
```

- `arcLength()` — возвращает периметр контура или длину кривой. Параметр `closed` задает флаг, который указывает, замкнута кривая или нет. Формат метода:

```
public static double arcLength(MatOfPoint2f curve,
                               boolean closed)
```

- `approxPolyDP()` — выполняет аппроксимацию контура `curve` с заданной точностью `epsilon` и записывает результат в матрицу `approxCurve`. Формат метода:

```
public static void approxPolyDP(MatOfPoint2f curve,
                                MatOfPoint2f approxCurve,
                                double epsilon, boolean closed)
```

- `isContourConvex()` — проверяет контур на выпуклость. Формат метода:

```
public static boolean isContourConvex(MatOfPoint contour)
```

- `convexHull()` — находит выпуклую оболочку для контура. Форматы метода:

```
public static void convexHull(MatOfPoint points, MatOfInt hull)
public static void convexHull(MatOfPoint points, MatOfInt hull,
                              boolean clockwise)
```

- `convexityDefects()` — находит дефекты выпуклости контура. В первом параметре указывается контур, во втором — результат выполнения метода `convexHull()`, а в третьем — матрица, в которую будет записан результат операции. Формат метода:

```
public static void convexityDefects(MatOfPoint contour,
              MatOfInt convexhull, MatOfInt4 convexityDefects)
```

- `fitEllipse()` — находит эллипс, который лучше всего подходит для множества точек. Возвращает повернутую на некоторый угол прямоугольную область,

в которую вписан эллипс. Обратите внимание: количество точек должно быть больше двух. Формат метода:

```
public static RotatedRect fitEllipse(MatOfPoint2f points)
```

Пример нахождения и отрисовки эллипса:

```
RotatedRect rect = Imgproc.fitEllipse(
            new MatOfPoint2f(contours.get(i).toArray()));
Imgproc.ellipse(img, rect, CvUtils.COLOR_WHITE, 1);
```

Чтобы преобразовать матрицу `MatOfPoint` в матрицу `MatOfPoint2f`, нужно с помощью метода `toArray()` получить массив точек, а затем передать этот массив конструктору класса `MatOfPoint2f`:

```
double len = Imgproc.arcLength(
            new MatOfPoint2f(contours.get(i).toArray()), true);
```

Найдем все контуры верхнего уровня, вычислим их площадь и обведем найденный контур белой рамкой (листинг 8.3). Результат выполнения кода из листинга 8.3 показан на рис. 8.3.

Листинг 8.3. Вычисление площади контура и обводка контура рамкой

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");

Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
Mat edges = new Mat();
Imgproc.Canny(imgGray, edges, 80, 200);
Mat edgesCopy = edges.clone(); // Создаем копию
ArrayList<MatOfPoint> contours = new ArrayList<MatOfPoint>();
Imgproc.findContours(edgesCopy, contours, new Mat(),
                     Imgproc.RETR_EXTERNAL,
                     Imgproc.CHAIN_APPROX_SIMPLE);
for (int i = 0, j = contours.size(); i < j; i++) {
   System.out.println(Imgproc.contourArea(contours.get(i)));
   Rect r = Imgproc.boundingRect(contours.get(i));
   System.out.println("boundingRect = " + r);
   double len = Imgproc.arcLength(
          new MatOfPoint2f(contours.get(i).toArray()), true);
   System.out.println("arcLength = " + len);
   Imgproc.rectangle(img, new Point(r.x, r.y),
                  new Point(r.x + r.width - 1, r.y + r.height - 1),
                  CvUtils.COLOR_WHITE);
}
```

```
CvUtilsFX.showImage(img, "boundingRect");
img.release();   imgGray.release();
edges.release(); edgesCopy.release();
```

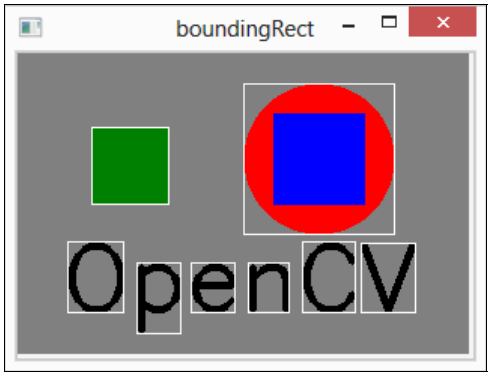


Рис. 8.3. Результат выполнения кода из листинга 8.3

8.1.5. Сравнение контуров

Чтобы сравнить два контура, нужно предварительно рассчитать *моменты* с помощью статического метода `moments()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static Moments moments(Mat array)
public static Moments moments(Mat array, boolean binaryImage)
```

В первом параметре указывается матрица с найденным контуром или изображение с одним каналом (8 битов или вещественные значения). Если в параметре `binaryImage` указать значение `true`, то все значения больше единицы обрабатываются как единица (параметр используется только для изображений). Метод `moments()` возвращает результат в виде экземпляра класса `Moments`. Инструкция импорта:

```
import org.opencv.imgproc.Moments;
```

Конструкторы класса:

```
Moments()
Moments(double m00, double m10, double m01, double m20, double m11,
        double m02, double m30, double m21, double m12, double m03)
Moments(double[] vals)
```

Получить значения можно с помощью следующих методов:

```
// Пространственные моменты
public double get_m00(); // Площадь контура
public double get_m01();
public double get_m02();
public double get_m03();
```

```
public double get_m10();
public double get_m11();
public double get_m12();
public double get_m20();
public double get_m21();
public double get_m30();
// Центральные моменты
public double get_mu02();
public double get_mu03();
public double get_mu11();
public double get_mu12();
public double get_mu20();
public double get_mu21();
public double get_mu30();
// Нормализованные центральные моменты
public double get_nu02();
public double get_nu03();
public double get_nu11();
public double get_nu12();
public double get_nu20();
public double get_nu21();
public double get_nu30();
```

Метод `get_m00()` вернет площадь контура. Чтобы задать значения, нужно использовать методы с приставкой `set` вместо `get`.

Вычислить центр масс на основе моментов можно следующим образом:

```
// Moments m = Imgproc.moments(contours.get(i));
double x_cm = m.get_m10() / m.get_m00();
double y_cm = m.get_m01() / m.get_m00();
```

Моменты зависят от масштаба и поворота контура. Чтобы избавиться от этой проблемы, используются *инвариантные моменты*, которые можно получить с помощью статического метода `HuMoments()` из класса `Imgproc`. Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static void HuMoments(Moments m, Mat hu)
```

В первом параметре указывается результат выполнения метода `moments()`, а во втором — матрица, в которую будет записан результат операции.

Пример вычисления моментов и центров масс приведен в листинге 8.4.

Листинг 8.4. Вычисление моментов и центров масс

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
```

```
CvUtilsFX.showImage(img, "Оригинал");

Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
Mat edges = new Mat();
Imgproc.Canny(imgGray, edges, 80, 200);
Mat edgesCopy = edges.clone(); // Создаем копию
ArrayList<MatOfPoint> contours = new ArrayList<MatOfPoint>();
Imgproc.findContours(edgesCopy, contours, new Mat(),
                     Imgproc.RETR_EXTERNAL,
                     Imgproc.CHAIN_APPROX_SIMPLE);
for (int i = 0, j = contours.size(); i < j; i++) {
   Rect r = Imgproc.boundingRect(contours.get(i));
   Moments m = Imgproc.moments(contours.get(i));
   // Пространственные моменты
   System.out.println("get_m00 = " + m.get_m00());
   System.out.println("get_m01 = " + m.get_m01());
   System.out.println("get_m02 = " + m.get_m02());
   System.out.println("get_m03 = " + m.get_m03());
   System.out.println("get_m10 = " + m.get_m10());
   System.out.println("get_m11 = " + m.get_m11());
   System.out.println("get_m12 = " + m.get_m12());
   System.out.println("get_m20 = " + m.get_m20());
   System.out.println("get_m21 = " + m.get_m21());
   System.out.println("get_m30 = " + m.get_m30());
   // Центральные моменты
   System.out.println("get_mu02 = " + m.get_mu02());
   System.out.println("get_mu03 = " + m.get_mu03());
   System.out.println("get_mu11 = " + m.get_mu11());
   System.out.println("get_mu12 = " + m.get_mu12());
   System.out.println("get_mu20 = " + m.get_mu20());
   System.out.println("get_mu21 = " + m.get_mu21());
   System.out.println("get_mu30 = " + m.get_mu30());
   // Нормализованные центральные моменты
   System.out.println("get_nu02 = " + m.get_nu02());
   System.out.println("get_nu03 = " + m.get_nu03());
   System.out.println("get_nu11 = " + m.get_nu11());
   System.out.println("get_nu12 = " + m.get_nu12());
   System.out.println("get_nu20 = " + m.get_nu20());
   System.out.println("get_nu21 = " + m.get_nu21());
   System.out.println("get_nu30 = " + m.get_nu30());
   // Центр масс
   double x_cm = m.get_m10() / m.get_m00();
   double y_cm = m.get_m01() / m.get_m00();
   Imgproc.circle(img, new Point(x_cm, y_cm), 3, CvUtils.COLOR_YELLOW,
                  Core.FILLED);
```

```
   // Инвариантные моменты
   Mat hu = new Mat();
   Imgproc.HuMoments(m, hu);
   System.out.println("HuMoments " + hu.dump());

   Imgproc.rectangle(img, new Point(r.x, r.y),
                  new Point(r.x + r.width - 1, r.y + r.height - 1),
                  CvUtils.COLOR_WHITE);
}
CvUtilsFX.showImage(img, "boundingRect");
img.release();   imgGray.release();
edges.release(); edgesCopy.release();
```

Сравнить два контура позволяет статический метод `matchShapes()` из класса `Imgproc`. Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static double matchShapes(Mat contour1, Mat contour2, int method,
                                 double parameter)
```

Метод сравнивает `contour1` с `contour2` на основе инвариантных моментов. В параметре `method` указывается способ сравнения (формулы смотрите в документации). Можно указать следующие константы из класса `Imgproc`:

```
public static final int CONTOURS_MATCH_I1
public static final int CONTOURS_MATCH_I2
public static final int CONTOURS_MATCH_I3
```

В версии 2.4 используются следующие константы:

```
public static final int CV_CONTOURS_MATCH_I1
public static final int CV_CONTOURS_MATCH_I2
public static final int CV_CONTOURS_MATCH_I3
```

Выведем значения констант:

```
System.out.println(Imgproc.CONTOURS_MATCH_I1); // 1
System.out.println(Imgproc.CONTOURS_MATCH_I2); // 2
System.out.println(Imgproc.CONTOURS_MATCH_I3); // 3
```

Параметр `parameter` зарезервирован и в настоящее время не используется. Метод возвращает результат сравнения в виде вещественного числа. Чем ближе это число к значению `0`, тем более похожи контуры.

Пример сравнения контуров приведен в листинге 8.5.

Листинг 8.5. Сравнение контуров

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
```

```
   return;
}
CvUtilsFX.showImage(img, "Оригинал");

Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
Mat edges = new Mat();
Imgproc.Canny(imgGray, edges, 80, 200);
Mat edgesCopy = edges.clone(); // Создаем копию
ArrayList<MatOfPoint> contours = new ArrayList<MatOfPoint>();
Imgproc.findContours(edgesCopy, contours, new Mat(),
                     Imgproc.RETR_EXTERNAL,
                     Imgproc.CHAIN_APPROX_SIMPLE);
MatOfPoint shape = new MatOfPoint();
if (contours.size() >= 4) {
   shape = contours.get(3);
   Imgproc.drawContours(img, contours, 3, CvUtils.COLOR_WHITE);
}
double min = Double.MAX_VALUE, value = 0;
int index = -1;
for (int i = 0, j = contours.size(); i < j; i++) {
   value = Imgproc.matchShapes(contours.get(i), shape,
                  Imgproc.CV_CONTOURS_MATCH_I3, 0);
   if (value < min) {
      min = value;
      index = i;
   }
   System.out.println("CV_CONTOURS_MATCH_I1: " + i + " " +
          Imgproc.matchShapes(contours.get(i), shape,
                  Imgproc.CV_CONTOURS_MATCH_I1, 0));
   System.out.println("CV_CONTOURS_MATCH_I2: " + i + " " +
          Imgproc.matchShapes(contours.get(i), shape,
                  Imgproc.CV_CONTOURS_MATCH_I2, 0));
   System.out.println("CV_CONTOURS_MATCH_I3: " + i + " " +
          Imgproc.matchShapes(contours.get(i), shape,
                  Imgproc.CV_CONTOURS_MATCH_I3, 0));
}
Rect r = Imgproc.boundingRect(contours.get(index));
Imgproc.rectangle(img, new Point(r.x, r.y),
               new Point(r.x + r.width - 1, r.y + r.height - 1),
               CvUtils.COLOR_WHITE);
System.out.println("Лучшее совпадение: индекс " + index +
                   " значение " + min);
CvUtilsFX.showImage(img, "Результат сравнения");
img.release();   imgGray.release();
edges.release(); edgesCopy.release(); shape.release();
```

8.2. Поиск объекта по цвету

При распознавании объектов на изображении цвет используется редко, т. к. времени на его обработку требуется в три раза больше. Поэтому в большинстве случаев на вход методы принимают изображения в градациях серого. Тем не менее, в ряде случаев без цвета не обойтись, — он, например, нужен для определения горящего сигнала светофора или для поиска объекта по цвету. Поиск по цвету также полезен при распознавании на игровом поле цветных фигур — когда, например, на белом фоне двигается красный мячик и нужно отслеживать его перемещения.

Проблема распознавания цвета на изображении заключается в том, что при различном освещении цвет может иметь разные оттенки. Если объект имеет блестящую поверхность, то дополнительно на нем может присутствовать блик от источника света. Человеческий глаз адаптируется к условиям освещения и автоматически изменяет баланс белого, а вот фотоаппараты и видеокамеры часто ошибаются при определении баланса белого, что приводит к искажению цвета на полученном с них изображении.

При поиске объекта по диапазону цветов следует учитывать, что цветовое пространство `RGB` не соответствует человеческому восприятию диапазона цветов. Для этого лучше подходит цветовое пространство `HSV` (включает каналы цветовой тон, насыщенность и яркость). Преобразовать цветовое пространство `BGR` в `HSV` можно так:

```
Imgproc.cvtColor(bgr, hsv, Imgproc.COLOR_BGR2HSV);
```

Найти объект по цвету позволяет статический метод `inRange()` из класса `Core`. Формат метода:

```
import org.opencv.core.Core;
public static void inRange(Mat src, Scalar lowerb, Scalar upperb,
                           Mat dst)
```

В первом параметре указывается исходное изображение, во втором — минимальные значения диапазона, а в третьем — максимальные значения диапазона. Результат, в виде двухцветного изображения, будет записан в матрицу `dst`. Пикселы, значения цветов которых попали в диапазон, станут белыми, а остальные — черными.

Пример поиска объекта по цвету приведен в листинге 8.6.

Листинг 8.6. Поиск объекта по цвету

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");
Mat hsv = new Mat();
Imgproc.cvtColor(img, hsv, Imgproc.COLOR_BGR2HSV);
```

```
Mat h = new Mat();
Core.extractChannel(hsv, h, 0);
Mat img2 = new Mat();
Core.inRange(h, new Scalar(40), new Scalar(80), img2);
CvUtilsFX.showImage(img2, "Зеленый");
Core.inRange(h, new Scalar(100), new Scalar(140), img2);
CvUtilsFX.showImage(img2, "Синий");
Core.inRange(hsv, new Scalar(0, 200, 200),
                  new Scalar(20, 256, 256), img2);
CvUtilsFX.showImage(img2, "Красный");
Core.inRange(hsv, new Scalar(0, 0, 0),
                  new Scalar(0, 0, 50), img2);
CvUtilsFX.showImage(img2, "Черный");
img.release(); img2.release(); hsv.release(); h.release();
```

8.3. Вычитание фона из текущего кадра

Если мы имеем два изображения и хотим найти различия второго изображения от первого, то можно просто вычесть второе изображение из первого. Одинаковые значения станут равны нулю, а значения больше нуля и будут искомыми отличиями. Вычислить абсолютную разность между двумя изображениями позволяет статический метод `absdiff()` из класса `Core`. Вначале вычисляется разница, затем знак отбрасывается и в конце проверяется выход за диапазон значений. Форматы метода:

```
import org.opencv.core.Core;
public static void absdiff(Mat src1, Scalar src2, Mat dst)
public static void absdiff(Mat src1, Mat src2, Mat dst)
```

Пример использования метода `absdiff()` приведен в листинге 8.7.

Листинг 8.7. Поиск отличий

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");
Mat img2 = img.clone();
Imgproc.circle(img2, new Point(100, 100), 20, CvUtils.COLOR_RED,
               Core.FILLED);
CvUtilsFX.showImage(img2, "Оригинал + круг");
Mat img3 = new Mat();
Core.absdiff(img2, img, img3);
CvUtilsFX.showImage(img3, "Разница");
Mat img4 = new Mat();
```

```
Imgproc.cvtColor(img3, img4, Imgproc.COLOR_BGR2GRAY);
Imgproc.threshold(img4, img4, 1, 255, Imgproc.THRESH_BINARY);
CvUtilsFX.showImage(img4, "threshold");
img.release(); img2.release(); img3.release(); img4.release();
```

При работе с камерами наружного наблюдения для поиска различий часто прибегают к методу вычитания фона из текущего кадра. Конечно же, при условии, что камера статична. Но если мы для этого попробуем воспользоваться методом `absdiff()`, то обнаружим множество проблем. Первая проблема заключается в том, что из-за цифровых шумов камеры два соседних кадра разные и без посторонних объектов. Вторую проблему представляет погода. Когда светит солнце, объекты отбрасывают тень, и эта тень медленно движется. Кроме того, солнце может заходить за тучи, и в результате освещенность резко меняется. Когда идет дождь или снег, различий становится очень много. При порывах ветра листья на деревьях будут шевелиться, а при сильных порывах может двигаться и сама камера. Помимо названных существует множество других проблем.

Использовать изображение фона на постоянной основе также нельзя. Во-первых, из-за смены времени суток, — очевидно, что фон ночью будет иным, нежели днем. Во-вторых, опять-таки из-за погоды. Ну и, наконец, объект может встать перед камерой и остаться там надолго. Например, автомобиль припарковался у обочины, и водитель куда-то надолго ушел. В последнем случае сам момент парковки нам интересен, а вот в дальнейшем, если с объектом ничего не происходит, автомобиль лучше сделать частью фона. Таким образом, изображение фона должно меняться с течением времени.

Для вычитания фона из текущего кадра видео в версии 3.3 предназначен класс `BackgroundSubtractorMOG2`. Инструкция импорта:

```
import org.opencv.video.BackgroundSubtractorMOG2;
```

Иерархия наследования:

```
Object — Algorithm — BackgroundSubtractor — BackgroundSubtractorMOG2
```

Создать объект класса `BackgroundSubtractorMOG2` в версии 3.3 позволяет статический метод `createBackgroundSubtractorMOG2()` из класса `Video`. Форматы метода:

```
import org.opencv.video.Video;
public static BackgroundSubtractorMOG2 createBackgroundSubtractorMOG2()
public static BackgroundSubtractorMOG2 createBackgroundSubtractorMOG2(
       int history, double varThreshold, boolean detectShadows)
```

Пример:

```
BackgroundSubtractorMOG2 bg = Video.createBackgroundSubtractorMOG2();
```

К сожалению, в версии 3.3.0 (и в 3.2.0) при работе с классом `BackgroundSubtractorMOG2` выводятся сообщения об ошибках. При этом вычитание фона из текущего кадра производится. Однако в результате получается не черно-белое изображение, а изображение в оттенках серого (если в параметре `detectShadows` указано значение `true`). В версии 2.4.13 таких проблем нет, поэтому примеры работы с классом `BackgroundSubtractorMOG2` мы будем рассматривать только для версии 2.4.

Создать экземпляр класса `BackgroundSubtractorMOG2` в версии 2.4 позволяют следующие конструкторы:

```
BackgroundSubtractorMOG2()
BackgroundSubtractorMOG2(int history, float varThreshold)
BackgroundSubtractorMOG2(int history, float varThreshold,
                         boolean bShadowDetection)
```

В версии 2.4 помимо класса `BackgroundSubtractorMOG2` доступен также класс `BackgroundSubtractorMOG`. Инструкция импорта:

```
import org.opencv.video.BackgroundSubtractorMOG; // Нет в 3.3
```

Иерархия наследования:

```
Object — Algorithm — BackgroundSubtractor — BackgroundSubtractorMOG
```

Конструкторы класса `BackgroundSubtractorMOG`:

```
BackgroundSubtractorMOG()
BackgroundSubtractorMOG(int history, int nmixtures,
                        double backgroundRatio)
BackgroundSubtractorMOG(int history, int nmixtures,
                        double backgroundRatio, double noiseSigma)
```

Задать фоновое изображение и получить результат сравнения текущего изображения с фоновым позволяет метод `apply()` из класса `BackgroundSubtractor`. Форматы метода:

```
public void apply(Mat image, Mat fgmask)
public void apply(Mat image, Mat fgmask, double learningRate)
```

В первом параметре указываем текущий кадр, а результат сравнения будет доступен через второй параметр в виде черно-белого изображения. Белым цветом будут обозначены отличающиеся области, а черным — совпадающие области.

Пример использования классов `BackgroundSubtractorMOG` и `BackgroundSubtractorMOG2` в версии 2.4 приведен в листинге 8.8.

Листинг 8.8. Вычитание фона из текущего кадра

```
// Только для версии 2.4
Mat img = Highgui.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");
Mat img2 = img.clone();
Core.circle(img2, new Point(100, 100), 20, CvUtils.COLOR_RED,
            Core.FILLED);
CvUtilsFX.showImage(img2, "Оригинал + круг");
```

```
BackgroundSubtractorMOG bg = new BackgroundSubtractorMOG();
Mat img3 = new Mat();
bg.apply(img, img3);
bg.apply(img2, img3);
CvUtilsFX.showImage(img3, "BackgroundSubtractorMOG");

BackgroundSubtractorMOG2 bg2 = new BackgroundSubtractorMOG2();
Mat img4 = new Mat();
bg2.apply(img, img4);
bg2.apply(img2, img4);
CvUtilsFX.showImage(img4, "BackgroundSubtractorMOG2");
img.release(); img2.release(); img3.release(); img4.release();
```

8.4. Поиск объекта по шаблону

Выполнить поиск объекта по шаблону позволяет статический метод `matchTemplate()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void matchTemplate(Mat image, Mat templ, Mat result,
                                 int method)
public static void matchTemplate(Mat image, Mat templ, Mat result,
                                 int method, Mat mask) // Нет в 2.4
```

В первом параметре указывается исходное изображение, а во втором — искомый шаблон. Результат будет записан в матрицу `result`. Эта матрица будет состоять из одного канала и содержать вещественные числа. Размер матрицы вычисляется следующим образом:

```
resultW = imageW - templW + 1
resultH = imageH - templH + 1
```

В параметре `method` указывается способ вычисления значений (формулы смотрите в документации). Можно указать следующие константы из класса `Imgproc`:

```
public static final int TM_SQDIFF
public static final int TM_SQDIFF_NORMED
public static final int TM_CCORR
public static final int TM_CCORR_NORMED
public static final int TM_CCOEFF
public static final int TM_CCOEFF_NORMED
```

Выведем значения констант:

```
System.out.println(Imgproc.TM_SQDIFF);        // 0
System.out.println(Imgproc.TM_SQDIFF_NORMED); // 1
System.out.println(Imgproc.TM_CCORR);         // 2
System.out.println(Imgproc.TM_CCORR_NORMED);  // 3
System.out.println(Imgproc.TM_CCOEFF);        // 4
System.out.println(Imgproc.TM_CCOEFF_NORMED); // 5
```

После выполнения метода лучшие совпадения можно найти как глобальные минимумы (при использовании константы `TM_SQDIFF`) или максимумы (при использовании констант `TM_CCORR` и `TM_CCOEFF`) с помощью метода `minMaxLoc()` из класса `Core` (см. *разд. 2.3*). Получить позицию минимальных и максимальных значений в матрице позволяют свойства `minLoc` и `maxLoc` из класса `MinMaxLocResult` (листинг 8.9).

Листинг 8.9. Поиск объекта по шаблону

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(img, "Оригинал");
Mat img2 = img.submat(new Rect(30, 120, 45, 60)).clone();
CvUtilsFX.showImage(img2, "Шаблон");

Mat result = new Mat();
Imgproc.matchTemplate(img, img2, result, Imgproc.TM_SQDIFF);
MinMaxLocResult r = Core.minMaxLoc(result);
System.out.println(r.minVal + " " + r.minLoc);

Imgproc.rectangle(img, r.minLoc, new Point(r.minLoc.x + img2.width() - 1,
                  r.minLoc.y + img2.height() - 1), CvUtils.COLOR_WHITE);
CvUtilsFX.showImage(img, "Результат поиска");

Mat result2 = new Mat();
Imgproc.matchTemplate(img, img2, result2, Imgproc.TM_CCOEFF);
MinMaxLocResult r2 = Core.minMaxLoc(result2);
System.out.println(r2.maxVal + " " + r2.maxLoc);

img.release(); img2.release();
result.release(); result2.release();
```

8.5. Поиск прямых линий и кругов

Выполнить поиск прямых линий и кругов можно с помощью *преобразований Хафа*. Для поиска линий по этому алгоритму используется статический метод `HoughLinesP()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void HoughLinesP(Mat image, Mat lines, double rho,
                               double theta, int threshold)
public static void HoughLinesP(Mat image, Mat lines, double rho,
                               double theta, int threshold,
                               double minLineLength, double maxLineGap)
```

В первом параметре указывается исходное черно-белое изображение (8 битов, один канал). Обратите внимание: это изображение может быть изменено внутри метода. Если нужно этого избежать, то следует передавать копию изображения. Результат поиска будет записан в матрицу `lines`. Каждый элемент итоговой матрицы содержит четыре канала. Значения в первых двух каналах описывают координаты начальной точки линии, а значения во вторых двух — координаты конечной точки.

Параметр `rho` задает разрешение по дистанции в пикселах, параметр `theta` — разрешение по углу в радианах, параметр `threshold` — пороговое значение, параметр `minLineLength` — ограничение по минимальной длине линии (значение по умолчанию: `0`), а параметр `maxLineGap` — максимальный промежуток между соединяемыми точками (значение по умолчанию: `0`).

Пример использования метода `HoughLinesP()` приведен в листинге 8.10.

Листинг 8.10. Поиск прямых линий

```
Mat img = new Mat(150, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.rectangle(img, new Point(20, 20), new Point(120, 70),
                  CvUtils.COLOR_GREEN, Core.FILLED);
Imgproc.line(img, new Point(20, 100), new Point(120, 100),
             CvUtils.COLOR_RED, 1);
Imgproc.line(img, new Point(20, 120), new Point(120, 120),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(150, 20), new Point(150, 100),
             CvUtils.COLOR_RED, 1);
Imgproc.line(img, new Point(170, 20), new Point(170, 100),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(200, 20), new Point(280, 100),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(280, 20), new Point(200, 100),
             CvUtils.COLOR_RED, 3);

Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
Mat edges = new Mat();
Imgproc.Canny(imgGray, edges, 80, 200);
CvUtilsFX.showImage(edges, "Canny");

Mat lines = new Mat();
Imgproc.HoughLinesP(edges, lines, 1, Math.toRadians(2), 20, 30, 0);
Mat result = new Mat(img.size(), CvType.CV_8UC3, CvUtils.COLOR_WHITE);
for (int i = 0, r = lines.rows(); i < r; i++) {
   for (int j = 0, c = lines.cols(); j < c; j++) {
      double[] line = lines.get(i, j);
      Imgproc.line(result, new Point(line[0], line[1]),
                   new Point(line[2], line[3]), CvUtils.COLOR_BLACK);
   }
}
```

```
CvUtilsFX.showImage(result, "Результат");
img.release();
imgGray.release();
edges.release();
result.release();
```

Для поиска прямых линий можно также воспользоваться статическим методом `HoughLines()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void HoughLines(Mat image, Mat lines, double rho,
                              double theta, int threshold)
public static void HoughLines(Mat image, Mat lines, double rho,
       double theta, int threshold, double srn, double stn,
       double min_theta, double max_theta) // Нет в 2.4
```

В версии 2.4 доступен также следующий формат метода:

```
public static void HoughLines(Mat image, Mat lines, double rho,
       double theta, int threshold, double srn, double stn)
```

Матрица `lines` с результатом операции содержит два канала. Значение в первом канале описывает расстояние от начала координат, а значение во втором канале — угол поворота линии в радианах.

Для поиска кругов предназначен статический метод `HoughCircles()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void HoughCircles(Mat image, Mat circles, int method,
                        double dp, double minDist)
public static void HoughCircles(Mat image, Mat circles, int method,
                        double dp, double minDist, double param1,
                        double param2, int minRadius, int maxRadius)
```

В первом параметре указывается исходное изображение в оттенках серого (8 битов, один канал). Результат поиска будет записан в матрицу `circles`. Каждый элемент итоговой матрицы содержит три канала. Значения в первых двух каналах описывают координаты центра круга, а значение в третьем — радиус. В параметре `method` указывается константа `HOUGH_GRADIENT` (другие методы не поддерживаются) из класса `Imgproc`. Формат:

```
public static final int HOUGH_GRADIENT // Нет в 2.4
```

В версии 2.4 используется константа `CV_HOUGH_GRADIENT`. Формат:

```
public static final int CV_HOUGH_GRADIENT
```

Параметр `dp` задает разрешение сумматора (`1` — одинаково с исходным изображением, `2` — половина ширины и высоты и т. д.), параметр `minDist` — минимальное расстояние между центрами обнаруженных кругов, параметр `minRadius` — мини-

мальный радиус круга (значение по умолчанию: 0), а параметр maxRadius — максимальный радиус круга (значение по умолчанию: 0).

Пример использования метода HoughCircles() приведен в листинге 8.11.

Листинг 8.11. Поиск кругов

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(imgGray, "Оригинал");

Mat circles = new Mat();
Imgproc.HoughCircles(imgGray, circles, Imgproc.HOUGH_GRADIENT,
                     2, imgGray.rows() / 4);
Mat result = new Mat(img.size(), CvType.CV_8UC3, CvUtils.COLOR_WHITE);
for (int i = 0, r = circles.rows(); i < r; i++) {
   for (int j = 0, c = circles.cols(); j < c; j++) {
      double[] circle = circles.get(i, j);
      Imgproc.circle(result, new Point(circle[0], circle[1]),
                    (int) circle[2], CvUtils.COLOR_BLACK);
   }
}
CvUtilsFX.showImage(result, "Результат");
img.release();
imgGray.release();
result.release();
```

8.6. Класс *LineSegmentDetector*

Выполнить поиск прямых линий в версии 3.3 позволяет класс LineSegmentDetector. Инструкция импорта:

```
import org.opencv.imgproc.LineSegmentDetector;
```

Иерархия наследования:

```
Object — Algorithm — LineSegmentDetector
```

Создать экземпляр класса LineSegmentDetector позволяет статический метод createLineSegmentDetector() из класса Imgproc. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static LineSegmentDetector createLineSegmentDetector()
```

```
public static LineSegmentDetector createLineSegmentDetector(
              int refine, double scale, double sigma_scale,
              double quant, double ang_th, double log_eps,
              double density_th, int n_bins)
```

Выполнить поиск линий можно с помощью метода `detect()`. Форматы метода:

```
public void detect(Mat image, Mat lines)
public void detect(Mat image, Mat lines, Mat width, Mat prec, Mat nfa)
```

В первом параметре указывается исходное изображение в оттенках серого (тип `CV_8UC1`). Координаты начальных и конечных точек найденных линий будут записаны в матрицу `lines` (тип `CV_32FC4`). Остальные параметры позволяют получить дополнительную информацию и по умолчанию принимают пустые матрицы. Например, если в параметре `width` указать ссылку на матрицу, то в нее будет записана информация о ширине линий.

Нарисовать найденные линии позволяет метод `drawSegments()`. Формат метода:

```
public void drawSegments(Mat image, Mat lines)
```

В первом параметре указывается изображение, на котором будут нарисованы линии, а во втором параметре — матрица с линиями, найденными с помощью метода `detect()`.

Пример поиска прямых линий приведен в листинге 8.12, а результат выполнения кода из листинга 8.12 показан на рис. 8.4.

Листинг 8.12. Поиск прямых линий

```
Mat img = new Mat(150, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Imgproc.rectangle(img, new Point(20, 20), new Point(120, 70),
                  CvUtils.COLOR_GREEN, Core.FILLED);
Imgproc.line(img, new Point(20, 100), new Point(120, 100),
             CvUtils.COLOR_RED, 1);
Imgproc.line(img, new Point(20, 120), new Point(120, 120),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(150, 20), new Point(150, 100),
             CvUtils.COLOR_RED, 1);
Imgproc.line(img, new Point(170, 20), new Point(170, 100),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(200, 20), new Point(280, 100),
             CvUtils.COLOR_RED, 3);
Imgproc.line(img, new Point(280, 20), new Point(200, 100),
             CvUtils.COLOR_RED, 3);

Mat imgGray = new Mat();
Imgproc.cvtColor(img, imgGray, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(imgGray, "Оригинал");
```

```
Mat lines = new Mat();
Mat width = new Mat();
Mat prec = new Mat();
Mat result = new Mat(img.size(), CvType.CV_8UC3, CvUtils.COLOR_WHITE);
LineSegmentDetector d = Imgproc.createLineSegmentDetector();
d.detect(imgGray, lines, width, prec, new Mat());
System.out.println(CvType.typeToString(lines.type())); // CV_32FC4
System.out.println(lines.size()); // 1x20
System.out.println(CvType.typeToString(width.type())); // CV_64FC1
System.out.println(width.size()); // 1x20
System.out.println(width.dump());
System.out.println(CvType.typeToString(prec.type()));  // CV_64FC1
System.out.println(prec.size());  // 1x20
System.out.println(prec.dump());

d.drawSegments(result, lines);
CvUtilsFX.showImage(result, "Результат");
img.release();
imgGray.release();
result.release();
```

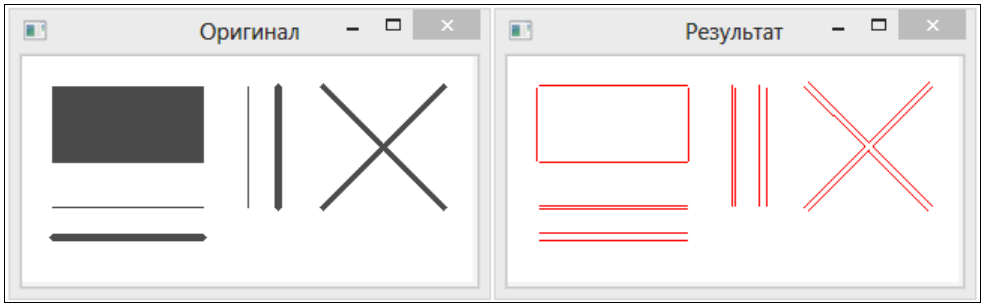
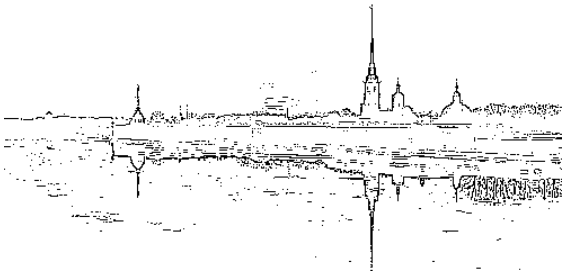


Рис. 8.4. Результат выполнения кода из листинга 8.12



ГЛАВА 9

Сегментация изображения

Сегментация изображения — это разделение изображения на несколько частей, внутри которых пикселы объединяются по некоторым признакам. Одной из областей применения сегментации является выделение объектов, например, для последующего вырезания объекта из изображения. Если вы работали с программой Photoshop, то знаете инструменты выделения **Магнитное лассо** и **Волшебная палочка**, так вот, например, метод `grabCut()`, который мы рассмотрим в этой главе, может использоваться как инструмент **Волшебная палочка**.

9.1. Метод *watershed()*

Статический метод `watershed()` из класса `Imgproc` выполняет сегментацию, используя алгоритм «*водораздела*». Формат метода:

```
import org.opencv.imgproc.Imgproc;
public static void watershed(Mat image, Mat markers)
```

В чем же заключается смысл алгоритма? Представьте себе изображение в оттенках серого. На нем есть светлые и темные области. Такое изображение можно представить в виде топографической карты с пиками и впадинами. Если мы начнем заливать изолированные впадины краской, то постепенно она будет доходить до небольших пиков и переливаться. Чтобы разные краски не смешивались, в этом месте ставится барьер. Процедура продолжается до тех пор, пока все пики не окажутся под краской. Результатом операции будут разделенные барьерами области, залитые красками разного цвета.

В OpenCV вначале нужно выделить интересующие области маркерами с разными индексами (например, линиями с разными оттенками серого). Результатом операции станут барьеры (имеют значение `-1`) между областями, обозначенными маркерами. Сами области будут обозначены индексами маркеров.

В первом параметре метод `watershed()` принимает исходное изображение из трех каналов с глубиной цвета 8 битов на канал, а во втором параметре — матрицу

с маркерами (один канал, глубина CV_32S). Результат выполнения метода будет записан в матрицу markers.

Пример использования метода watershed() приведен в листинге 9.1, а результат выполнения кода из листинга 9.1 показан на рис. 9.1.

Листинг 9.1. Метод watershed()

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");
// Рисуем маркеры
Mat mask = new Mat(img.size(), CvType.CV_8UC1, new Scalar(0));
Imgproc.line(mask, new Point(30, 30), new Point(130, 30),
             new Scalar(255), 5);
Imgproc.line(mask, new Point(60, 80), new Point(90, 80),
             new Scalar(255), 5);
Imgproc.line(mask, new Point(180, 80), new Point(210, 80),
             new Scalar(255), 5);
Imgproc.line(mask, new Point(238, 60), new Point(238, 75),
             new Scalar(255), 5);
CvUtilsFX.showImage(mask, "Маска с маркерами");
// Находим контуры маркеров
ArrayList<MatOfPoint> contours = new ArrayList<MatOfPoint>();
Imgproc.findContours(mask, contours, new Mat(),
                     Imgproc.RETR_CCOMP,
                     Imgproc.CHAIN_APPROX_SIMPLE);
// Отрисовываем контуры разными оттенками серого
Mat markers = new Mat(img.size(), CvType.CV_32SC1, new Scalar(0));
int color = 80;
for (int i = 0, j = contours.size(); i < j; i++) {
   Imgproc.drawContours(markers, contours, i, Scalar.all(color), 1);
   color += 20;
}
Imgproc.watershed(img, markers);
// Отображаем результат
Mat result = new Mat();
markers.convertTo(result, CvType.CV_8U);
CvUtilsFX.showImage(result, "Результат");
img.release();     mask.release();
markers.release(); result.release();
```

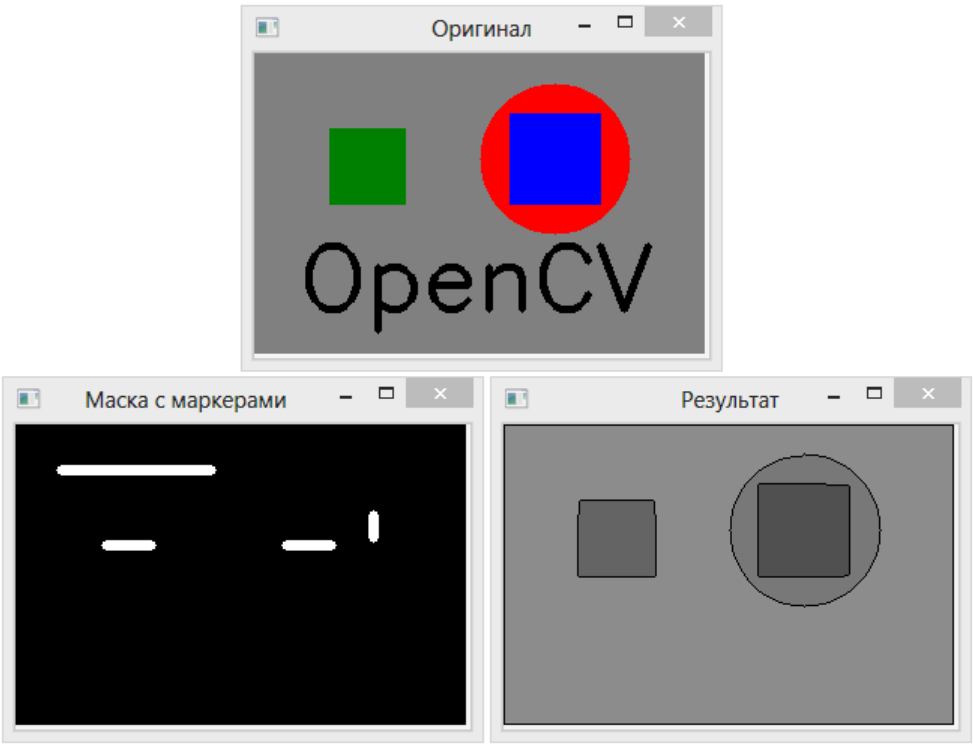


Рис. 9.1. Результат выполнения кода из листинга 9.1

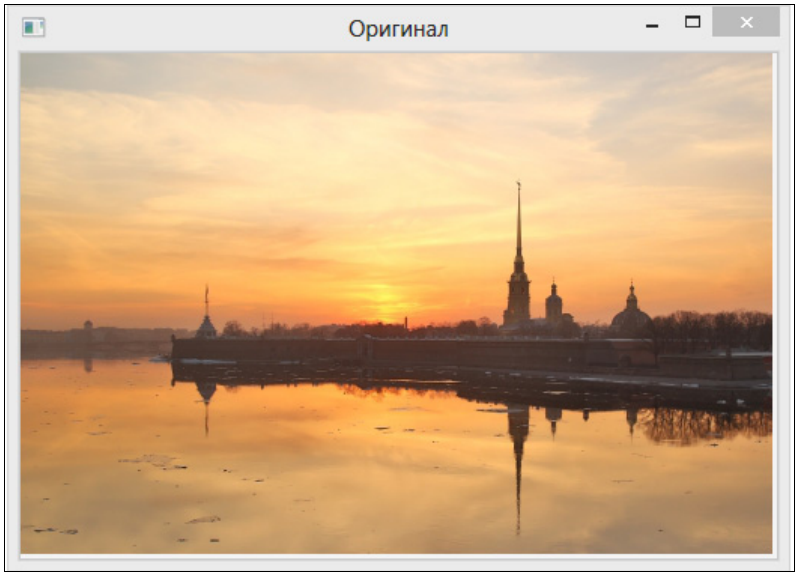
9.2. Метод *pyrMeanShiftFiltering()*

Статический метод `pyrMeanShiftFiltering()` из класса `Imgproc` группирует области с близкими признаками. Форматы метода:

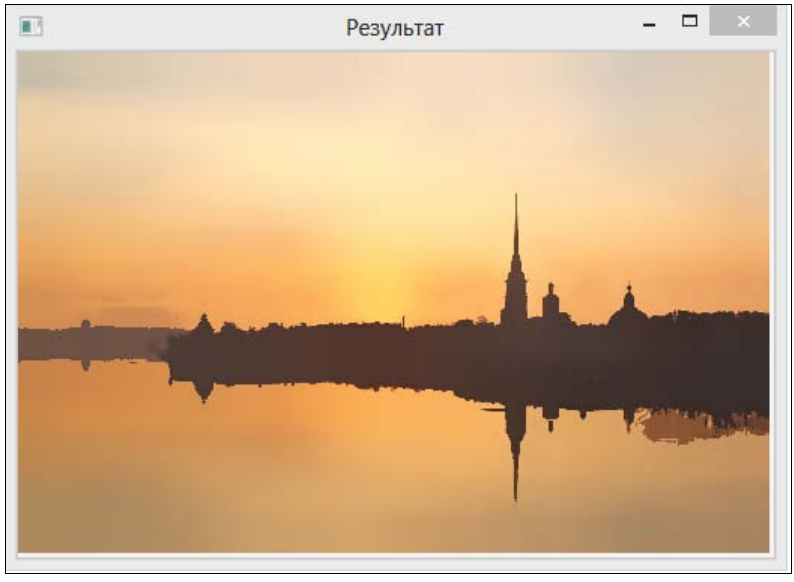
```
import org.opencv.imgproc.Imgproc;
import org.opencv.core.TermCriteria;
public static void pyrMeanShiftFiltering(Mat src, Mat dst,
                                         double sp, double sr)
public static void pyrMeanShiftFiltering(Mat src, Mat dst,
                                         double sp, double sr,
                                         int maxLevel, TermCriteria termcrit)
```

В первом параметре указывается исходное изображение, содержащее три канала по 8 битов, а во втором — матрица, в которую будет записан результат операции. Параметр `sp` задает радиус пространственного окна, а параметр `sr` — радиус цветового окна. В параметре `maxLevel` можно дополнительно указать максимальное количество уровней гауссовой пирамиды, а в параметре `termcrit` — критерии завершения итераций в виде объекта класса `TermCriteria`.

Пример использования метода `pyrMeanShiftFiltering()` приведен в листинге 9.2, а результат выполнения кода из листинга 9.2 показан на рис. 9.2.



а



б

Рис. 9.2. Результат выполнения кода из листинга 9.2: *а* — до обработки; *б* — после обработки

Листинг 9.2. Метод `pyrMeanShiftFiltering()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
```

```
CvUtilsFX.showImage(img, "Оригинал");
Mat result = new Mat();
Imgproc.pyrMeanShiftFiltering(img, result, 20, 50, 1,
   new TermCriteria(TermCriteria.MAX_ITER + TermCriteria.EPS, 5, 1));
CvUtilsFX.showImage(result, "Результат");
img.release();   result.release();
```

9.3. Метод *floodFill()*

Статический метод `floodFill()` из класса `Imgproc` выполняет заливку однородных или градиентных областей каким-либо сплошным цветом. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static int floodFill(Mat image, Mat mask, Point seedPoint,
                            Scalar newVal)
public static int floodFill(Mat image, Mat mask, Point seedPoint,
                            Scalar newVal, Rect rect, Scalar loDiff,
                            Scalar upDiff, int flags)
```

В первом параметре указывается матрица с исходным изображением (с одним или тремя каналами). Обратите внимание: по умолчанию результат операции будет записан в ту же самую матрицу. Во втором параметре можно указать маску (один канал, 8 битов). Размеры маски должны быть на 2 пиксела шире и выше исходного изображения. Параметр `seedPoint` задает начальную точку внутри заливаемой области, а параметр `newVal` — новый цвет. В объект, указанный в параметре `rect`, будут записаны координаты и размеры прямоугольной области, ограничивающей область заливки. В параметре `loDiff` можно указать минимальное значение яркости или цвета, а в параметре `upDiff` — максимальное. Параметр `flags` задает дополнительные флаги. Если указано значение `4` (значение по умолчанию), то учитываются только четыре соседних пиксела, а если `8` — то восемь соседних. Следующие восемь битов задают значение для маски: от `1` (значение по умолчанию) до `255`. Записать значение `255` можно так:

```
4 | (255 << 8)
```

Дополнительно можно указать следующие флаги (константы из класса `Imgproc`):

- `FLOODFILL_FIXED_RANGE` — если флаг установлен, то параметры `loDiff` и `upDiff` задают разницу между начальным и конечным пикселами. В противном случае они задают разницу для соседних пикселов. Формат:

  ```
  public static final int FLOODFILL_FIXED_RANGE
  ```

- `FLOODFILL_MASK_ONLY` — если флаг установлен, то метод не станет изменять исходное изображение. Результат будет записан в матрицу, указанную в параметре `mask`. Формат:

  ```
  public static final int FLOODFILL_MASK_ONLY
  ```

Пример заливки одноцветной области приведен в листинге 9.3, а результат выполнения кода из листинга 9.3 показан на рис. 9.3.

Листинг 9.3. Метод `floodFill()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");
Imgproc.floodFill(img, new Mat(), new Point(20, 20),
                  new Scalar(255, 255, 255));
CvUtilsFX.showImage(img, "Результат");
img.release();
```

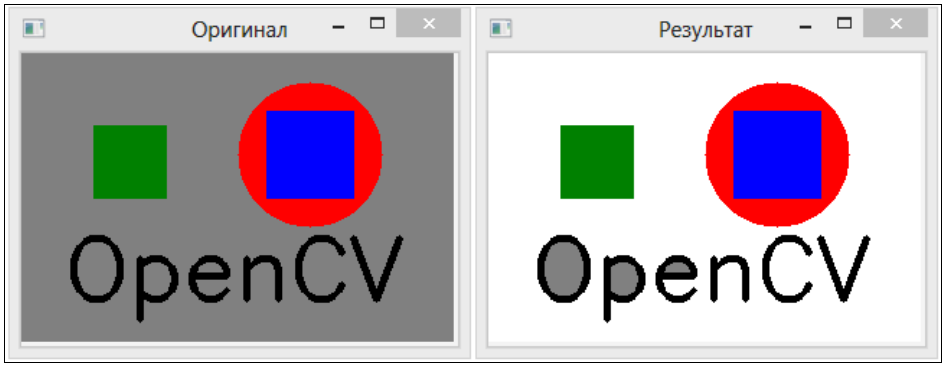


Рис. 9.3. Результат выполнения кода из листинга 9.3

Пример заливки градиентной области приведен в листинге 9.4, а результат выполнения кода из листинга 9.4 показан на рис. 9.4.

Листинг 9.4. Заливка градиентной области

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto9.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(img, "Оригинал");
Mat mask = new Mat(img.rows() + 2, img.cols() + 2,
                   CvType.CV_8UC1, new Scalar(0));
Rect r = new Rect();
Imgproc.floodFill(img, mask, new Point(0, 0),
                  new Scalar(255, 255, 255), r,
                  new Scalar(20), new Scalar(40), 4 | (255 << 8));
CvUtilsFX.showImage(mask, "Маска");
CvUtilsFX.showImage(img, "Результат");
```

```
System.out.println(r);
img.release();
mask.release();
```

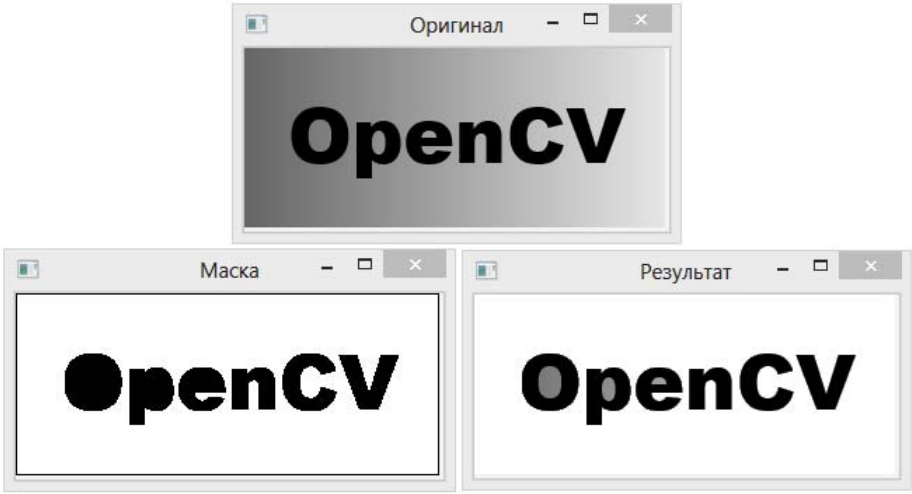


Рис. 9.4. Результат выполнения кода из листинга 9.4

9.4. Метод *grabCut()*

Статический метод `grabCut()` из класса `Imgproc` позволяет отделить объект от фона. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void grabCut(Mat img, Mat mask, Rect rect, Mat bgdModel,
                           Mat fgdModel, int iterCount)
public static void grabCut(Mat img, Mat mask, Rect rect, Mat bgdModel,
                           Mat fgdModel, int iterCount, int mode)
```

В первом параметре указывается исходное изображение, содержащее три канала по 8 битов, а во втором — матрица с маской (один канал, 8 битов). Параметр `rect` задает положение и размеры прямоугольной области, вне которой пикселы отмечаются как фоновые (`GC_BGD`). Используется параметр `rect` только при указании флага `GC_INIT_WITH_RECT` в параметре `mode`. Параметр `bgdModel` задает временный массив для фоновой модели, а параметр `fgdModel` — для модели переднего плана. Изменять значения этих массивов вручную не следует. Их нужно передавать при повторных вызовах метода при установленном флаге `GC_EVAL`. Параметр `iterCount` задает количество итераций. В параметре `mode` можно указать следующие режимы (константы из класса `Imgproc`):

- `GC_INIT_WITH_RECT` — состояние и маска инициализируется с помощью прямоугольной области, указанной в параметре `rect`. Формат:

  ```
  public static final int GC_INIT_WITH_RECT
  ```

- `GC_INIT_WITH_MASK` — состояние инициализируется с помощью маски, указанной в параметре `mask`. Формат:

```
public static final int GC_INIT_WITH_MASK
```

- `GC_EVAL` — повторное выполнение. Формат:

```
public static final int GC_EVAL
```

Результат операции будет доступен через маску. Внутри маски пикселы отмечаются с помощью следующих значений:

- `GC_BGD` — пиксел принадлежит фону. Формат:

```
public static final int GC_BGD
```

- `GC_FGD` — пиксел принадлежит переднему плану. Формат:

```
public static final int GC_FGD
```

- `GC_PR_BGD` — пиксел возможно принадлежит фону. Формат:

```
public static final int GC_PR_BGD
```

- `GC_PR_FGD` — пиксел возможно принадлежит переднему плану. Формат:

```
public static final int GC_PR_FGD
```

Выведем значения констант:

```
System.out.println(Imgproc.GC_BGD);    // 0
System.out.println(Imgproc.GC_FGD);    // 1
System.out.println(Imgproc.GC_PR_BGD); // 2
System.out.println(Imgproc.GC_PR_FGD); // 3
```

Пример использования метода `grabCut()` приведен в листинге 9.5, а результат выполнения кода из листинга 9.5 показан на рис. 9.5.

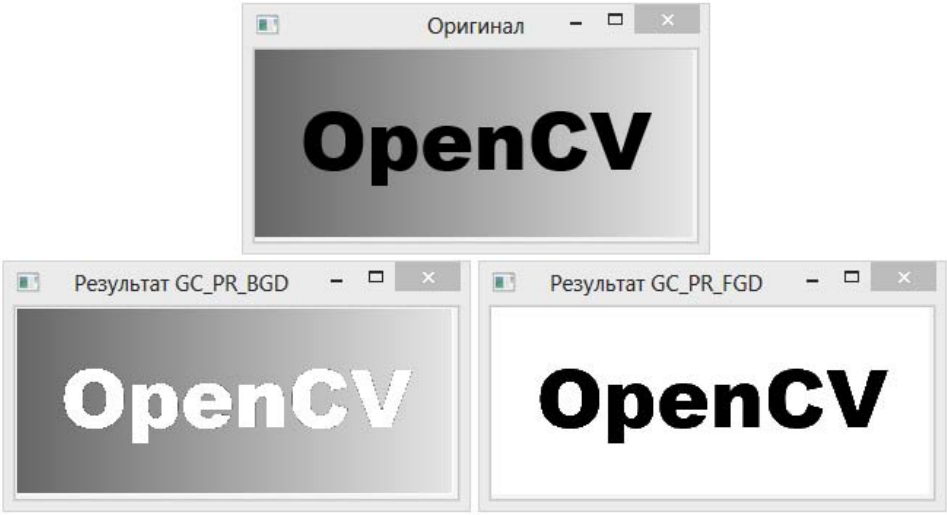


Рис. 9.5. Результат выполнения кода из листинга 9.5

Листинг 9.5. Метод `grabCut()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto9.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CvUtilsFX.showImage(img, "Оригинал");

Mat bgdModel = new Mat();
Mat fgdModel = new Mat();

// Инициализация с помощью прямоугольной области
Mat mask = new Mat();
// Rect rect = new Rect(150, 10, 140, 100);
Rect rect = new Rect(1, 1, img.width() - 2, img.height() - 2);
Imgproc.grabCut(img, mask, rect, bgdModel, fgdModel, 1,
                Imgproc.GC_INIT_WITH_RECT);
/*
// Инициализация с помощью маски
Mat mask = new Mat(img.size(), CvType.CV_8UC1,
                   new Scalar(Imgproc.GC_BGD));
Imgproc.rectangle(mask, new Point(1, 1),
                  new Point(img.width() - 2, img.height() - 2),
                  new Scalar(Imgproc.GC_PR_FGD), Core.FILLED);
Imgproc.grabCut(img, mask, new Rect(), bgdModel, fgdModel, 1,
                Imgproc.GC_INIT_WITH_MASK);
*/
// Повторный вызов
Imgproc.grabCut(img, mask, new Rect(), bgdModel, fgdModel, 1,
                Imgproc.GC_EVAL);

Mat maskPR_FGD = new Mat();
Core.compare(mask, new Scalar(Imgproc.GC_PR_FGD), maskPR_FGD,
             Core.CMP_EQ);
Mat resultPR_FGD = new Mat(img.rows(), img.cols(), CvType.CV_8UC3,
                           CvUtils.COLOR_WHITE);
img.copyTo(resultPR_FGD, maskPR_FGD);
CvUtilsFX.showImage(resultPR_FGD, "Результат GC_PR_FGD");

Mat maskPR_BGD = new Mat();
Core.compare(mask, new Scalar(Imgproc.GC_PR_BGD), maskPR_BGD,
             Core.CMP_EQ);
Mat resultPR_BGD = new Mat(img.rows(), img.cols(), CvType.CV_8UC3,
                           CvUtils.COLOR_WHITE);
img.copyTo(resultPR_BGD, maskPR_BGD);
CvUtilsFX.showImage(resultPR_BGD, "Результат GC_PR_BGD");
```

```
img.release();          mask.release();
maskPR_BGD.release();   maskPR_FGD.release();
resultPR_BGD.release(); resultPR_FGD.release();
```

9.5. Метод *kmeans()*

Статический метод `kmeans()` из класса `Core` выполняет кластеризацию, используя алгоритм K-средних. Форматы метода:

```
import org.opencv.core.Core;
import org.opencv.core.TermCriteria;
public static double kmeans(Mat data, int K, Mat bestLabels,
                            TermCriteria criteria, int attempts,
                            int flags)
public static double kmeans(Mat data, int K, Mat bestLabels,
                            TermCriteria criteria, int attempts,
                            int flags, Mat centers)
```

В первом параметре указывается матрица с исходными данными (глубина `CV_32F`). Чтобы преобразовать изображение в такую матрицу, нужно воспользоваться методом `reshape()` для изменения структуры матрицы, а затем преобразовать ее к типу `CV_32F`:

```
Mat data = img.reshape(1, img.rows() * img.cols() * img.channels());
data.convertTo(data, CvType.CV_32F, 1.0 / 255);
```

В параметре `K` указывается нужное количество кластеров в результате выполнения операции. Индексы найденных кластеров будут записаны в матрицу `bestLabels` (глубина `CV_32S`). Параметр `criteria` задает критерии завершения итераций в виде объекта класса `TermCriteria`. Количество повторов указывается в параметре `attempts`. В параметре `flags` можно указать следующие флаги (константы из класса `Core`):

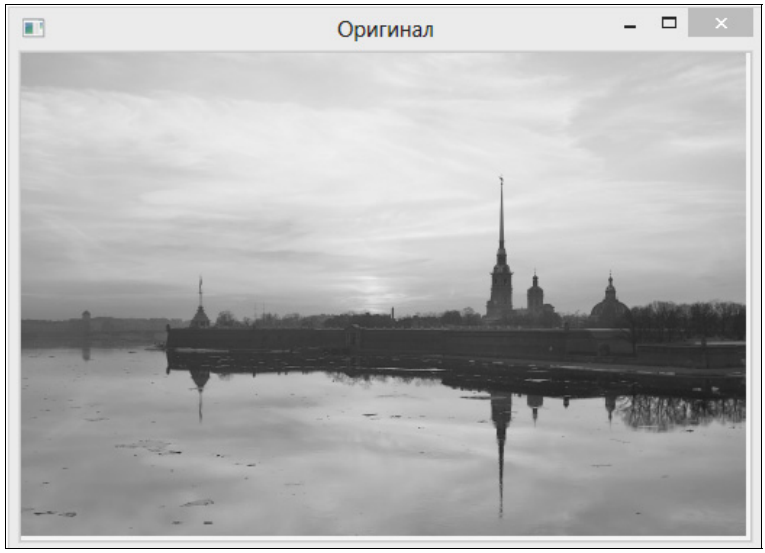
```
public static final int KMEANS_PP_CENTERS
public static final int KMEANS_RANDOM_CENTERS
public static final int KMEANS_USE_INITIAL_LABELS
```

В матрицу `centers` будут записаны центры кластеров. Количество элементов в этой матрице совпадает с количеством кластеров, указанных в параметре `K`.

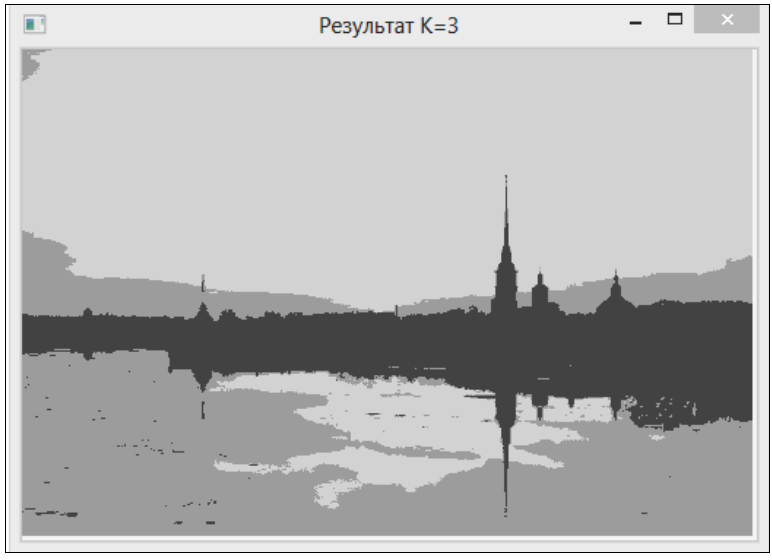
Пример сегментации изображения с помощью метода `kmeans()` приведен в листинге 9.6, а результат выполнения кода из листинга 9.6 показан на рис. 9.6.

Листинг 9.6. Метод `kmeans()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
```



а

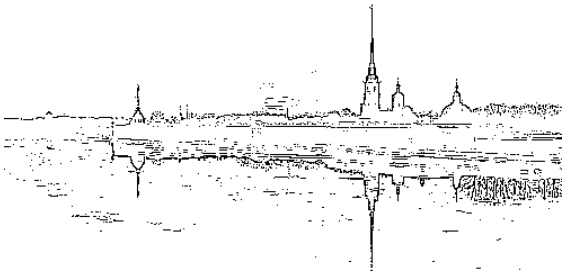


б

Рис. 9.6. Результат выполнения кода из листинга 9.6: *а* — до обработки; *б* — после обработки

```
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(img, "Оригинал");

Mat data = img.reshape(1, img.rows() * img.cols() * img.channels());
data.convertTo(data, CvType.CV_32F, 1.0 / 255);
Mat bestLabels = new Mat();
Mat centers = new Mat();
TermCriteria criteria = new TermCriteria(
                TermCriteria.MAX_ITER + TermCriteria.EPS, 10, 1);
```

```
int K = 3;
Core.kmeans(data, K, bestLabels, criteria, 5,
            Core.KMEANS_RANDOM_CENTERS, centers);

Mat colors = new Mat();
centers.t().convertTo(colors, CvType.CV_8U, 255);
Mat lut = new Mat(1, 256, CvType.CV_8UC1, new Scalar(0));
colors.copyTo(
         new Mat(lut, new Range(0, 1), new Range(0, colors.cols())));

Mat result = bestLabels.reshape(img.channels(), img.rows());
result.convertTo(result, CvType.CV_8U);
Core.LUT(result, lut, result);

CvUtilsFX.showImage(result, "Результат K=" + K);
img.release();        data.release();     result.release();
bestLabels.release(); centers.release();
colors.release();     lut.release();
```



ГЛАВА 10

Поиск особых точек

Чтобы иметь возможность найти на изображении какой-либо объект, нужно предварительно выделить на нем характерные признаки. Такими признаками являются *особые точки* — например, углы, центры окружностей и т. д. После нахождения особых точек нужно вычислить *дескрипторы* — векторы, характеризующие окрестности особых точек. Сравнивая дескрипторы двух точек, можно найти соответствие особой точки на одном изображении с особой точкой на другом.

10.1. Поиск углов

Наиболее характерной особой точкой изображения является угол. Обычно углы делят на L-образные, T-образные, Y-образные, X-образные и в виде стрелки.

10.1.1. Детектор углов Харриса

Статический метод `cornerHarris()` из класса `Imgproc` реализует детектор углов Харриса. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void cornerHarris(Mat src, Mat dst, int blockSize,
                                int ksize, double k)
public static void cornerHarris(Mat src, Mat dst, int blockSize,
                                int ksize, double k, int borderType)
```

В первом параметре указывается исходное изображение, содержащее один канал (глубина `CV_8U` или `CV_32F`), а во втором параметре — ссылка на матрицу, в которую будет записан результат операции (глубина `CV_32F`). Параметр `blockSize` задает размер соседства, а параметр `ksize` — размер ядра фильтра для метода `Sobel()`. В параметре `k` можно указать коэффициент. Маленькие значения коэффициента (например, `0.04`) подходят для обнаружения углов, а большие — для обнаружения границ. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`. Углы на изображении можно найти как локальные максимумы в матрице `dst`.

Пример использования метода `cornerHarris()` приведен в листинге 10.1, а результат выполнения кода из листинга 10.1 показан на рис. 10.1.

Листинг 10.1. Метод `cornerHarris()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(img, "Оригинал");

Mat dst = new Mat();
Imgproc.cornerHarris(img, dst, 2, 3, 0.04);

MinMaxLocResult m = Core.minMaxLoc(dst);
Imgproc.threshold(dst, dst, m.maxVal * 0.01, 1.0, Imgproc.THRESH_BINARY);

CvUtilsFX.showImage(dst, "Результат");
img.release();  dst.release();
```

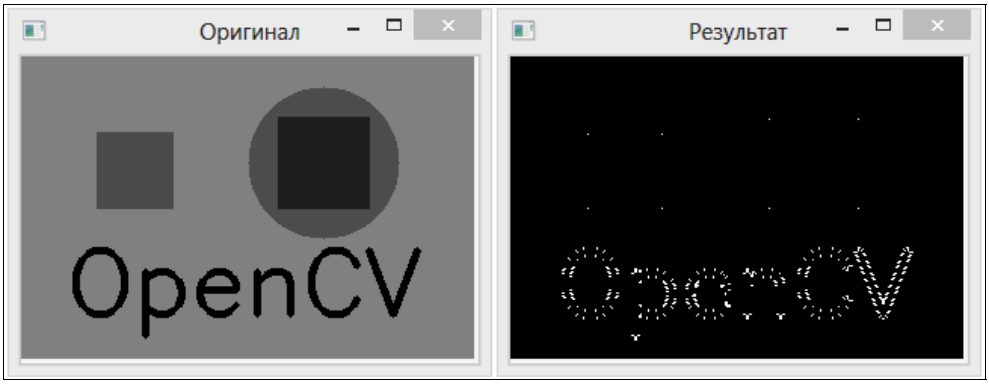


Рис. 10.1. Результат выполнения кода из листинга 10.1

10.1.2. Метод *cornerMinEigenVal()*

Статический метод `cornerMinEigenVal()` из класса `Imgproc` реализует детектор углов Shi-Tomasi. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void cornerMinEigenVal(Mat src, Mat dst, int blockSize)
public static void cornerMinEigenVal(Mat src, Mat dst, int blockSize,
                                      int ksize)
public static void cornerMinEigenVal(Mat src, Mat dst, int blockSize,
                                      int ksize, int borderType)
```

В первом параметре указывается исходное изображение, содержащее один канал (глубина `CV_8U` или `CV_32F`), а во втором параметре — ссылка на матрицу, в которую будет записан результат операции (глубина `CV_32F`). Параметр `blockSize` задает размер соседства, а параметр `ksize` — размер ядра фильтра для метода `Sobel()`. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`. Углы на изображении можно найти как локальные максимумы в матрице `dst`.

Получить более подробную информацию об угле позволяет статический метод `cornerEigenValsAndVecs()` из класса `Imgproc`. Форматы метода:

```
public static void cornerEigenValsAndVecs(Mat src, Mat dst,
                              int blockSize, int ksize)
public static void cornerEigenValsAndVecs(Mat src, Mat dst,
                              int blockSize, int ksize, int borderType)
```

Назначение параметров точно такое же, как и у одноименных параметров метода `cornerMinEigenVal()`. Различие состоит в структуре матрицы с результатом. Каждый элемент этой матрицы содержит шесть каналов: `lambda_1`, `lambda_2`, `x_1`, `y_1`, `x_2`, `y_2`. Метод `cornerMinEigenVal()` вычисляет лишь минимальное значение из `lambda_1` и `lambda_2`.

Пример использования методов `cornerEigenValsAndVecs()` и `cornerMinEigenVal()` приведен в листинге 10.2, а результат выполнения кода из листинга 10.2 показан на рис. 10.2.

Листинг 10.2. Методы `cornerEigenValsAndVecs()` и `cornerMinEigenVal()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(img, "Оригинал");

Mat dst = new Mat();
Imgproc.cornerMinEigenVal(img, dst, 2, 3);

Mat dst2 = new Mat();
Imgproc.cornerEigenValsAndVecs(img, dst2, 2, 3);

MinMaxLocResult m = Core.minMaxLoc(dst);
System.out.println(dst2.channels()); // 6
double[] maxDst = dst.get((int) m.maxLoc.y, (int) m.maxLoc.x);
System.out.println(Arrays.toString(maxDst));
double[] maxDst2 = dst2.get((int) m.maxLoc.y, (int) m.maxLoc.x);
System.out.println(Arrays.toString(maxDst2));
```

```
Imgproc.threshold(dst, dst, m.maxVal * 0.01, 1.0, Imgproc.THRESH_BINARY);
dst.convertTo(dst, CvType.CV_8U, 255);

CvUtilsFX.showImage(dst, "Результат");
img.release();  dst.release();
```

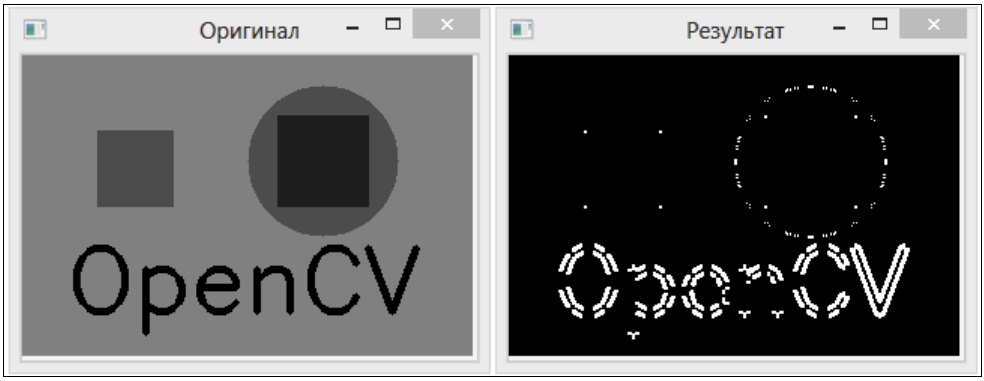


Рис. 10.2. Результат выполнения кода из листинга 10.2

10.1.3. Метод *preCornerDetect()*

Найти углы на изображении можно также с помощью статического метода `preCornerDetect()` из класса `Imgproc`. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void preCornerDetect(Mat src, Mat dst, int ksize)
public static void preCornerDetect(Mat src, Mat dst, int ksize,
                                   int borderType)
```

В первом параметре указывается исходное изображение, содержащее один канал (глубина `CV_8U` или `CV_32F`), а во втором параметре — ссылка на матрицу, в которую будет записан результат операции (глубина `CV_32F`). Параметр `ksize` задает размер ядра фильтра для метода `Sobel()`. В параметре `borderType` можно указать тип рамки вокруг изображения (см. *разд. 4.9*). По умолчанию используется значение `BORDER_DEFAULT`. Углы на изображении можно найти как локальные максимумы в матрице `dst`.

Пример использования метода `preCornerDetect()` приведен в листинге 10.3, а результат выполнения кода из листинга 10.3 показан на рис. 10.3.

Листинг 10.3. Метод `preCornerDetect()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
```

```
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(img, "Оригинал");

Mat dst = new Mat();
Imgproc.preCornerDetect(img, dst, 3);

MinMaxLocResult m = Core.minMaxLoc(dst);
Imgproc.threshold(dst, dst, m.maxVal * 0.01, 1.0, Imgproc.THRESH_BINARY);
dst.convertTo(dst, CvType.CV_8U, 255);

CvUtilsFX.showImage(dst, "Результат");
img.release();  dst.release();
```

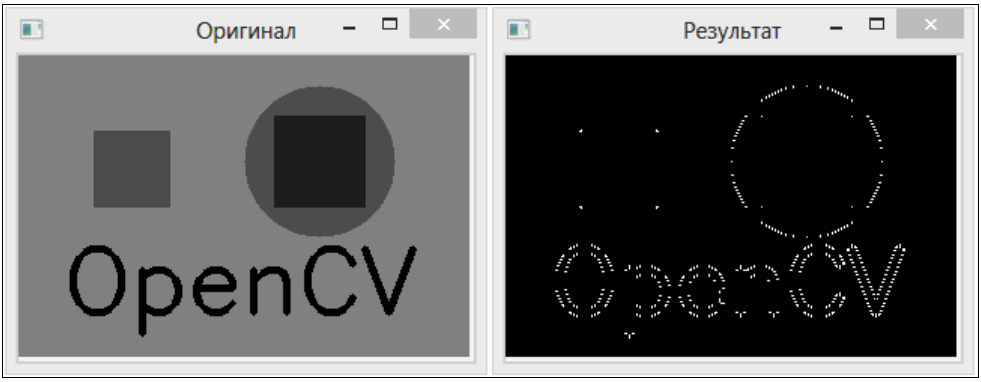


Рис. 10.3. Результат выполнения кода из листинга 10.3

10.1.4. Метод *goodFeaturesToTrack()*

Статический метод `goodFeaturesToTrack()` из класса `Imgproc` позволяет найти в изображении указанное количество «сильных» углов. Форматы метода:

```
import org.opencv.imgproc.Imgproc;
public static void goodFeaturesToTrack(Mat image, MatOfPoint corners,
                      int maxCorners, double qualityLevel,
                      double minDistance)
public static void goodFeaturesToTrack(Mat image, MatOfPoint corners,
                      int maxCorners, double qualityLevel,
                      double minDistance, Mat mask, int blockSize,
                      boolean useHarrisDetector, double k)
```

В первом параметре указывается исходное изображение, содержащее один канал (глубина `CV_8U` или `CV_32F`), а во втором параметре — ссылка на матрицу, в которую будет записан результат операции. Параметр `maxCorners` задает максимальное количество углов (в первую очередь возвращаются самые «сильные» углы), параметр `qualityLevel` — минимальное приемлемое качество угла, а параметр `minDistance` —

минимальное евклидово расстояние между углами. В параметре `mask` можно указать маску. Параметр `blockSize` задает размер соседства (значение по умолчанию: `3`). Если в параметре `useHarrisDetector` указано значение `true`, то для поиска углов будет использоваться метод `cornerHarris()`, а если `false` (значение по умолчанию) — то метод `cornerMinEigenVal()`. Параметр `k` задает коэффициент для метода `cornerHarris()` (значение по умолчанию: `0.04`).

Пример использования метода `goodFeaturesToTrack()` приведен в листинге 10.4, а результат выполнения листинга 10.4 показан на рис. 10.4.

Листинг 10.4. Метод `goodFeaturesToTrack()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(img, "Оригинал");

MatOfPoint corners = new MatOfPoint();
Imgproc.goodFeaturesToTrack(img, corners, 50, 0.01, 10);

double[] v = null;
for (int i = 0, r = corners.rows(); i < r; i++) {
   for (int j = 0, c = corners.cols(); j < c; j++) {
      v = corners.get(i, j);
      Imgproc.circle(img, new Point(v[0], v[1]), 2,
                     CvUtils.COLOR_WHITE, Core.FILLED);
   }
}
CvUtilsFX.showImage(img, "Результат");
img.release();
```

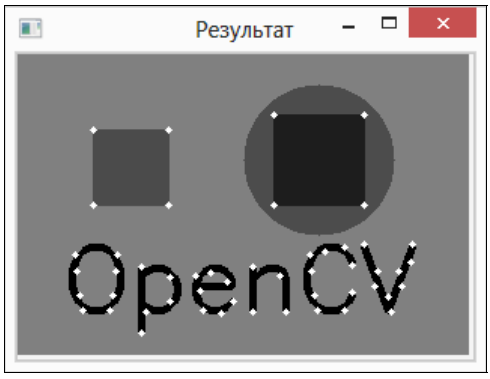


Рис. 10.4. Результат выполнения кода из листинга 10.4

10.1.5. Уточнение местоположения углов

Статический метод `cornerSubPix()` из класса `Imgproc` позволяет уточнить местоположение углов с субпиксельной точностью. Формат метода:

```
import org.opencv.imgproc.Imgproc;
import org.opencv.core.TermCriteria;
public static void cornerSubPix(Mat image, Mat corners, Size winSize,
                                 Size zeroZone, TermCriteria criteria)
```

В версии 2.4 используется следующий формат метода:

```
public static void cornerSubPix(Mat image, MatOfPoint2f corners,
                                 Size winSize, Size zeroZone,
                                 TermCriteria criteria)
```

В первом параметре указывается исходное изображение, а во втором — матрица с координатами углов (например, найденных с помощью метода `goodFeaturesToTrack()`). В матрицу `corners` будут записаны уточненные координаты углов. Параметр `winSize` задает половину длины стороны окна поиска. Если указан размер `Size(5, 5)`, то размеры окна поиска вычисляются так:

```
size = Size(5 * 2 + 1, 5 * 2 + 1)
```

В параметре `zeroZone` можно задать размеры области в середине окна поиска, внутри которой вычисления не производятся. Если указать значение `Size(-1, -1)`, то такой области нет. Параметр `criteria` задает критерии завершения итераций в виде объекта класса `TermCriteria`.

Пример использования метода `cornerSubPix()` приведен в листинге 10.5.

Листинг 10.5. Метод `cornerSubPix()`

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
CvUtilsFX.showImage(img, "Оригинал");

MatOfPoint corners = new MatOfPoint();
Imgproc.goodFeaturesToTrack(img, corners, 50, 0.01, 10,
                            new Mat(), 3, true, 0.04);
Mat imgCopy = img.clone();
double[] v = null;
for (int i = 0, r = corners.rows(); i < r; i++) {
   for (int j = 0, c = corners.cols(); j < c; j++) {
      v = corners.get(i, j);
```

```
      Imgproc.circle(imgCopy, new Point(v[0], v[1]), 2,
                  CvUtils.COLOR_WHITE, Core.FILLED);
   }
}
CvUtilsFX.showImage(imgCopy, "Результат goodFeaturesToTrack");

// Уточнение положения углов
MatOfPoint2f corners2 = new MatOfPoint2f(corners.toArray());
TermCriteria criteria = new TermCriteria(
      TermCriteria.MAX_ITER + TermCriteria.EPS, 100, 0.01);
Imgproc.cornerSubPix(img, corners2, new Size(5, 5),
                     new Size(-1, -1), criteria);
for (int i = 0, r = corners2.rows(); i < r; i++) {
   for (int j = 0, c = corners2.cols(); j < c; j++) {
      v = corners2.get(i, j);
      Imgproc.circle(img, new Point(v[0], v[1]), 2,
                     CvUtils.COLOR_WHITE, Core.FILLED);
   }
}
CvUtilsFX.showImage(img, "Результат cornerSubPix");
img.release(); imgCopy.release();
```

10.2. Поиск ключевых точек

Углы являются хорошими особыми точками, но только до тех пор, пока мы не изменим масштаб изображения, после чего характеристики углов могут измениться. Кроме того, углы бывают скругленными, а некоторые изображения могут вообще не содержать углов. Например, посмотрите на узоры, расположенные на крыльях бабочки, — большинство из них имеют округлые формы. В этом случае особой точкой может стать центр окружности или центр масс. Таким образом, ключевыми точками могут быть как углы, так и другие особенности изображения. Для работы с ключевыми точками в OpenCV предназначены классы из пакета `org.opencv.features2d`, которые мы сейчас и рассмотрим.

10.2.1. Класс *KeyPoint*

Класс `KeyPoint` описывает свойства ключевой точки. Инструкция импорта для версии 3.3:

```
import org.opencv.core.KeyPoint;
```

Инструкция импорта для версии 2.4:

```
import org.opencv.features2d.KeyPoint;
```

Конструкторы класса:

```
KeyPoint()
KeyPoint(float x, float y, float size)
```

```
KeyPoint(float x, float y, float size, float angle)
KeyPoint(float x, float y, float size, float angle, float response)
KeyPoint(float x, float y, float size, float angle, float response,
         int octave)
KeyPoint(float x, float y, float size, float angle, float response,
         int octave, int class_id)
```

Свойства класса:

- `pt` — координаты ключевой точки. Формат:

  ```
  public Point pt
  ```

- `size` — диаметр области. Формат:

  ```
  public float size
  ```

- `angle` — ориентация ключевой точки в градусах (имеет значение `-1`, если не применяется). Формат:

  ```
  public float angle
  ```

- `response` — значение, по которому отбираются самые «сильные» ключевые точки. Формат:

  ```
  public float response
  ```

- `octave` — слой пирамиды с ключевой точкой. Формат:

  ```
  public int octave
  ```

- `class_id` — идентификатор объекта. Формат:

  ```
  public int class_id
  ```

Пример:

```
KeyPoint p = new KeyPoint();
System.out.println(p);
// KeyPoint [pt={0.0, 0.0}, size=0.0, angle=-1.0,
// response=0.0, octave=0, class_id=-1]
KeyPoint p2 = new KeyPoint(20, 30, 1, 45, 0, 0, -1);
System.out.println(p2);
// KeyPoint [pt={20.0, 30.0}, size=1.0, angle=45.0,
// response=0.0, octave=0, class_id=-1]
System.out.println(p2.pt);       // {20.0, 30.0}
System.out.println(p2.size);     // 1.0
System.out.println(p2.angle);    // 45.0
System.out.println(p2.response); // 0.0
System.out.println(p2.octave);   // 0
System.out.println(p2.class_id); // -1
```

10.2.2. Класс *MatOfKeyPoint*

Класс `MatOfKeyPoint` реализует матрицу с ключевыми точками. Каждый элемент такой матрицы содержит семь каналов. Инструкция импорта:

```
import org.opencv.core.MatOfKeyPoint;
```

Иерархия наследования:

```
Object — Mat — MatOfKeyPoint
```

Конструкторы класса:

```
MatOfKeyPoint()
MatOfKeyPoint(KeyPoint... a)
MatOfKeyPoint(Mat m)
```

Пример:

```
MatOfKeyPoint m = new MatOfKeyPoint(new KeyPoint(20, 30, 1),
                                    new KeyPoint(10, 80, 2));
System.out.println(m.dump());
/*
[20, 30, 1, -1, 0, 0, -1;
 10, 80, 2, -1, 0, 0, -1]*/
System.out.println(m.channels()); // 7
System.out.println(m.type());     // 53
System.out.println(m.size());     // 1x2
```

Методы класса `MatOfKeyPoint`:

- `fromArray()` — заполняет матрицу значениями из массива. Формат метода:

  ```
  public void fromArray(KeyPoint... a)
  ```

  Пример:

  ```
  MatOfKeyPoint m = new MatOfKeyPoint();
  m.fromArray(new KeyPoint(20, 30, 1), new KeyPoint(10, 80, 2));
  System.out.println(m.dump());
  /*
  [20, 30, 1, -1, 0, 0, -1;
   10, 80, 2, -1, 0, 0, -1]*/
  ```

- `fromList()` — заполняет матрицу значениями из списка. Формат метода:

  ```
  public void fromList(List<KeyPoint> lkp)
  ```

  Пример:

  ```
  ArrayList<KeyPoint> list = new ArrayList<KeyPoint>();
  Collections.addAll(list, new KeyPoint(20, 30, 1),
                           new KeyPoint(10, 80, 2));
  MatOfKeyPoint m = new MatOfKeyPoint();
  m.fromList(list);
  System.out.println(m.dump());
  ```

```
/*
[20, 30, 1, -1, 0, 0, -1;
 10, 80, 2, -1, 0, 0, -1]*/
```

- `toArray()` — возвращает массив со значениями из матрицы. Формат метода:

```
public KeyPoint[] toArray()
```

Пример:

```
MatOfKeyPoint m = new MatOfKeyPoint(new KeyPoint(20, 30, 1),
                                    new KeyPoint(10, 80, 2));
KeyPoint[] arr = m.toArray();
System.out.println(Arrays.toString(arr));
/*
[KeyPoint [pt={20.0, 30.0}, size=1.0, angle=-1.0, response=0.0,
 octave=0, class_id=-1], KeyPoint [pt={10.0, 80.0}, size=2.0,
  angle=-1.0, response=0.0, octave=0, class_id=-1]]
*/
```

- `toList()` — возвращает список со значениями из матрицы. Формат метода:

```
public List<KeyPoint> toList()
```

Пример:

```
MatOfKeyPoint m = new MatOfKeyPoint(new KeyPoint(20, 30, 1),
                                    new KeyPoint(10, 80, 2));
List<KeyPoint> list = m.toList();
System.out.println(list);
/*
[KeyPoint [pt={20.0, 30.0}, size=1.0, angle=-1.0, response=0.0,
 octave=0, class_id=-1], KeyPoint [pt={10.0, 80.0}, size=2.0,
  angle=-1.0, response=0.0, octave=0, class_id=-1]]
*/
```

Для создания матрицы из списка можно также воспользоваться статическим методом `vector_KeyPoint_to_Mat()` из класса `Converters`. Формат метода:

```
import org.opencv.utils.Converters;
public static Mat vector_KeyPoint_to_Mat(List<KeyPoint> kps)
```

Выполнить обратную операцию позволяет статический метод `Mat_to_vector_KeyPoint()` из класса `Converters`. Формат метода:

```
public static void Mat_to_vector_KeyPoint(Mat m, List<KeyPoint> kps)
```

Пример:

```
ArrayList<KeyPoint> list = new ArrayList<KeyPoint>();
Collections.addAll(list, new KeyPoint(20, 30, 1),
                         new KeyPoint(10, 80, 2));
Mat m = Converters.vector_KeyPoint_to_Mat(list);
System.out.println(m.dump());
```

```
/*
[20, 30, 1, -1, 0, 0, -1;
 10, 80, 2, -1, 0, 0, -1]*/
System.out.println(m.channels()); // 7
System.out.println(m.type());     // 54
System.out.println(m.size());     // 1x2
ArrayList<KeyPoint> list2 = new ArrayList<KeyPoint>();
Converters.Mat_to_vector_KeyPoint(m, list2);
System.out.println(list2);
/*
[KeyPoint [pt={20.0, 30.0}, size=1.0, angle=-1.0, response=0.0,
 octave=0, class_id=-1], KeyPoint [pt={10.0, 80.0}, size=2.0,
  angle=-1.0, response=0.0, octave=0, class_id=-1]]
*/
```

Обратите внимание на тип возвращаемой матрицы: класс `MatOfKeyPoint` возвращает матрицу типа `CV_32FC(7)`, а методы из класса `Converters` работают с матрицами типа `CV_64FC(7)`:

```
System.out.println(CvType.CV_32FC(7)); // 53
System.out.println(CvType.CV_64FC(7)); // 54
```

10.2.3. Отрисовка ключевых точек

Нарисовать найденные ключевые точки на изображении позволяет статический метод `drawKeypoints()` из класса `Features2d`. Форматы метода:

```
import org.opencv.features2d.Features2d;
public static void drawKeypoints(Mat image, MatOfKeyPoint keypoints,
                                 Mat outImage)
public static void drawKeypoints(Mat image, MatOfKeyPoint keypoints,
                                 Mat outImage, Scalar color, int flags)
```

В первом параметре указывается исходное изображение, во втором — матрица с ключевыми точками, а в третьем — матрица, в которую будет записан результат операции. В параметре `color` можно задать цвет ключевых точек. Если параметр `color` не задан, или указано значение `Scalar.all(-1)`, то ключевые точки рисуются разными цветами. В параметре `flags` можно указать следующие флаги (константы из класса `Features2d`):

- `0` (значение по умолчанию) — для матрицы `outImage` вызывается метод `create()`, потом копируется изображение из матрицы `image` и на получившемся изображении рисуются ключевые точки;
- `DRAW_OVER_OUTIMG` — ключевые точки рисуются на существующем изображении в матрице `outImage` (метод `create()` не вызывается). Формат:

  ```
  public static final int DRAW_OVER_OUTIMG
  ```

- `NOT_DRAW_SINGLE_POINTS` — единичные ключевые точки не рисуются. Формат:

  ```
  public static final int NOT_DRAW_SINGLE_POINTS
  ```

- `DRAW_RICH_KEYPOINTS` — при рисовании будут учитываться размеры и ориентация ключевой точки (рисуется круг, внутри которого прямой линией из центра круга показывается ориентация). Формат:

```
public static final int DRAW_RICH_KEYPOINTS
```

Выведем значения констант:

```
System.out.println(Features2d.DRAW_OVER_OUTIMG);       // 1
System.out.println(Features2d.NOT_DRAW_SINGLE_POINTS); // 2
System.out.println(Features2d.DRAW_RICH_KEYPOINTS );   // 4
```

Пример:

```
Mat m = new Mat(50, 300, CvType.CV_8UC3, CvUtils.COLOR_WHITE);
MatOfKeyPoint kp = new MatOfKeyPoint(new KeyPoint(30, 30, 30, 90),
                                     new KeyPoint(120, 20, 20, 10),
                                     new KeyPoint(80, 30, 15, 45));
Mat m2 = new Mat();
Features2d.drawKeypoints(m, kp, m2);
CvUtilsFX.showImage(m2, "Без флага");
Mat m3 = new Mat(50, 300, CvType.CV_8UC3, CvUtils.COLOR_GRAY);
Features2d.drawKeypoints(m, kp, m3, CvUtils.COLOR_BLUE,
                         Features2d.DRAW_OVER_OUTIMG);
CvUtilsFX.showImage(m3, "DRAW_OVER_OUTIMG");
Mat m4 = m.clone();
Features2d.drawKeypoints(m, kp, m4, Scalar.all(-1),
                         Features2d.DRAW_RICH_KEYPOINTS);
CvUtilsFX.showImage(m4, "DRAW_RICH_KEYPOINTS");
```

Результат отрисовки ключевых точек показан на рис. 10.5.

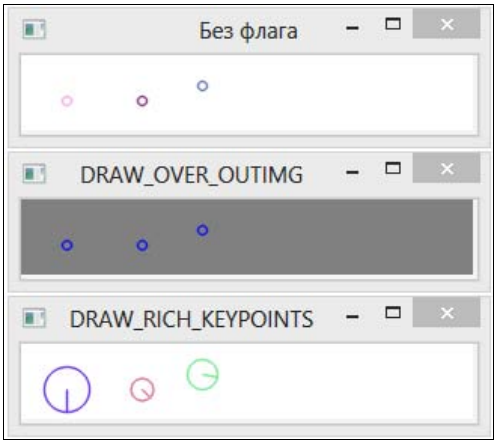


Рис. 10.5. Варианты отрисовки ключевых точек

10.2.4. Класс *FeatureDetector*

Класс `FeatureDetector` описывает методы поиска ключевых точек на изображении с использованием различных алгоритмов. Инструкция импорта:

```
import org.opencv.features2d.FeatureDetector;
```

Создать объект позволяет статический метод `create()`. Формат метода:

```
public static FeatureDetector create(int detectorType)
```

В параметре `detectorType` можно указать следующие алгоритмы поиска (статические константы из класса `FeatureDetector`):

```
public static final int SIFT             // Только в версии 2.4
public static final int PYRAMID_SIFT     // Только в версии 2.4
public static final int GRID_SIFT        // Только в версии 2.4
public static final int SURF             // Только в версии 2.4
public static final int PYRAMID_SURF     // Только в версии 2.4
public static final int GRID_SURF        // Только в версии 2.4
public static final int GFTT
public static final int PYRAMID_GFTT
public static final int GRID_GFTT
public static final int HARRIS
public static final int PYRAMID_HARRIS
public static final int GRID_HARRIS
public static final int FAST
public static final int PYRAMID_FAST
public static final int GRID_FAST
public static final int STAR             // Только в версии 2.4
public static final int PYRAMID_STAR     // Только в версии 2.4
public static final int GRID_STAR        // Только в версии 2.4
public static final int ORB
public static final int PYRAMID_ORB
public static final int GRID_ORB
public static final int BRISK
public static final int PYRAMID_BRISK
public static final int GRID_BRISK
public static final int DENSE            // Только в версии 2.4
public static final int PYRAMID_DENSE    // Только в версии 2.4
public static final int GRID_DENSE       // Только в версии 2.4
public static final int SIMPLEBLOB
public static final int PYRAMID_SIMPLEBLOB
public static final int GRID_SIMPLEBLOB
public static final int MSER
public static final int PYRAMID_MSER
public static final int GRID_MSER
public static final int AKAZE            // Только в версии 3.3
public static final int PYRAMID_AKAZE    // Только в версии 3.3
public static final int GRID_AKAZE       // Только в версии 3.3
```

Обратите внимание: на очень популярные алгоритмы `SIFT` и `SURF` (ускоренная версия `SIFT`) имеется патент, поэтому их нельзя использовать по свободной лицензии OpenCV в коммерческих целях. В OpenCV версии 3.3.0 эти алгоритмы вообще недоступны.

Алгоритмы `GFTT` и `HARRIS` используют метод `goodFeaturesToTrack()` (см. *разд. 10.1.4*). Алгоритм `GFTT` реализует детектор углов Shi-Tomasi (см. *разд. 10.1.2*), а алгоритм `HARRIS` — детектор углов Харриса (см. *разд. 10.1.1*).

Алгоритмы, названия которых начинаются с `PYRAMID_`, используют для поиска ключевых точек гауссовы пирамиды (см. *разд. 7.5*), а алгоритмы, названия которых начинаются с `GRID_`, — сетку размером `4x4`.

Найти ключевые точки позволяет метод `detect()` из класса `FeatureDetector`. Форматы метода:

```
public void detect(Mat image, MatOfKeyPoint keypoints)
public void detect(Mat image, MatOfKeyPoint keypoints, Mat mask)
```

В параметре `image` указывается исходное изображение, в параметре `keypoints` — матрица, в которую будет записан результат, а в параметре `mask` — маска.

Пример поиска ключевых точек различными алгоритмами приведен в листинге 10.6, а результат выполнения кода из листинга 10.6 показан на рис. 10.6.

Листинг 10.6. Поиск ключевых точек

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);

MatOfKeyPoint kp = new MatOfKeyPoint();
FeatureDetector fd = FeatureDetector.create(FeatureDetector.AKAZE);
fd.detect(img, kp);
Mat result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE,
                         Features2d.DRAW_RICH_KEYPOINTS);
CvUtilsFX.showImage(result, "AKAZE");

fd = FeatureDetector.create(FeatureDetector.GFTT);
kp = new MatOfKeyPoint();
fd.detect(img, kp);
result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "GFTT");
```

```
fd = FeatureDetector.create(FeatureDetector.HARRIS);
kp = new MatOfKeyPoint();
fd.detect(img, kp);
result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "HARRIS");

fd = FeatureDetector.create(FeatureDetector.FAST);
kp = new MatOfKeyPoint();
fd.detect(img, kp);
result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "FAST");

fd = FeatureDetector.create(FeatureDetector.ORB);
kp = new MatOfKeyPoint();
fd.detect(img, kp);
result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "ORB");

fd = FeatureDetector.create(FeatureDetector.BRISK);
kp = new MatOfKeyPoint();
fd.detect(img, kp);
result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "BRISK");

fd = FeatureDetector.create(FeatureDetector.SIMPLEBLOB);
kp = new MatOfKeyPoint();
fd.detect(img, kp);
result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE,
                         Features2d.DRAW_RICH_KEYPOINTS);
CvUtilsFX.showImage(result, "SIMPLEBLOB");

fd = FeatureDetector.create(FeatureDetector.PYRAMID_MSER);
kp = new MatOfKeyPoint();
fd.detect(img, kp);
result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE,
                         Features2d.DRAW_RICH_KEYPOINTS);
CvUtilsFX.showImage(result, "PYRAMID_MSER");
img.release();  result.release();
```

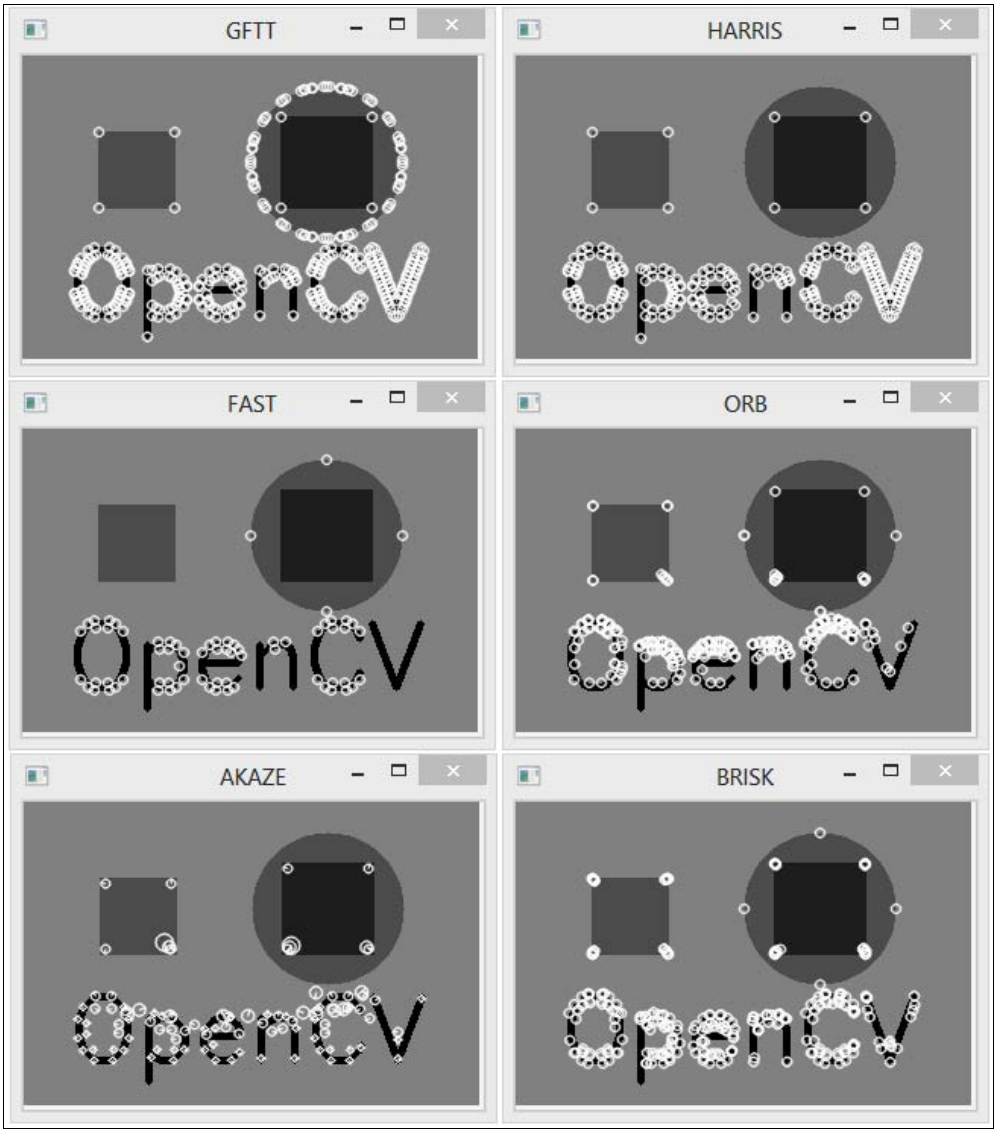


Рис. 10.6. Результат выполнения кода из листинга 10.6

10.3. Сравнение ключевых точек

Найденные в изображении ключевые точки можно сравнить с ключевыми точками другого изображения. Для этого вначале нужно вычислить дескрипторы с помощью методов из класса `DescriptorExtractor`, а затем выполнить сравнение дескрипторов с помощью методов из класса `DescriptorMatcher` и выбрать из результатов сравнения самые похожие точки.

10.3.1. Класс *DescriptorExtractor*

Класс `DescriptorExtractor` описывает методы для вычисления *дескрипторов* — векторов, кодирующих геометрию локальной окрестности вокруг ключевой точки. Инструкция импорта:

```
import org.opencv.features2d.DescriptorExtractor;
```

Создать объект позволяет статический метод `create()`. Формат метода:

```
public static DescriptorExtractor create(int extractorType)
```

В параметре `extractorType` можно указать следующие алгоритмы вычисления дескрипторов (статические константы из класса `DescriptorExtractor`):

```
public static final int SIFT             // Только в версии 2.4
public static final int OPPONENT_SIFT    // Только в версии 2.4
public static final int SURF             // Только в версии 2.4
public static final int OPPONENT_SURF    // Только в версии 2.4
public static final int ORB
public static final int OPPONENT_ORB
public static final int BRIEF            // Только в версии 2.4
public static final int OPPONENT_BRIEF   // Только в версии 2.4
public static final int BRISK
public static final int OPPONENT_BRISK
public static final int FREAK            // Только в версии 2.4
public static final int OPPONENT_FREAK   // Только в версии 2.4
public static final int AKAZE            // Только в версии 3.3
public static final int OPPONENT_AKAZE   // Только в версии 3.3
```

Обратите внимание: на алгоритмы `SIFT` и `SURF` имеется патент, поэтому их нельзя использовать по свободной лицензии OpenCV в коммерческих целях. В OpenCV версии 3.3.0 эти алгоритмы вообще недоступны.

Вычислить дескрипторы позволяет метод `compute()` из класса `DescriptorExtractor`. Формат метода:

```
public void compute(Mat image, MatOfKeyPoint keypoints, Mat descriptors)
```

В параметре `image` указывается исходное изображение, в параметре `keypoints` — матрица с ключевыми точками, а в параметре `descriptors` — матрица, в которую будет записан результат. Получить размер дескриптора позволяет метод `descriptorSize()`. Формат метода:

```
public int descriptorSize()
```

Пример поиска ключевых точек и вычисления дескрипторов приведен в листинге 10.7.

Листинг 10.7. Поиск ключевых точек и вычисление дескрипторов

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
```

```
   return;
}
MatOfKeyPoint kp_ORB = new MatOfKeyPoint();
FeatureDetector fd_ORB = FeatureDetector.create(FeatureDetector.ORB);
fd_ORB.detect(img, kp_ORB);
DescriptorExtractor de_ORB = DescriptorExtractor.create(
                              DescriptorExtractor.ORB);
Mat descriptors_ORB = new Mat();
de_ORB.compute(img, kp_ORB, descriptors_ORB);
System.out.println(de_ORB.descriptorSize());     // 32
System.out.println(descriptors_ORB.size());      // 32x262
System.out.println(descriptors_ORB.channels());  // 1
Mat result_ORB = new Mat();
Features2d.drawKeypoints(img, kp_ORB, result_ORB,
                         CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result_ORB, "ORB");

FeatureDetector fd_AKAZE = FeatureDetector.create(FeatureDetector.AKAZE);
MatOfKeyPoint kp_AKAZE = new MatOfKeyPoint();
fd_AKAZE.detect(img, kp_AKAZE);
DescriptorExtractor de_AKAZE = DescriptorExtractor.create(
                                DescriptorExtractor.AKAZE);
Mat descriptors_AKAZE = new Mat();
de_AKAZE.compute(img, kp_AKAZE, descriptors_AKAZE);
System.out.println(de_AKAZE.descriptorSize());      // 61
System.out.println(descriptors_AKAZE.size());       // 61x106
System.out.println(descriptors_AKAZE.channels());   // 1
Mat result = new Mat();
Features2d.drawKeypoints(img, kp_AKAZE, result,
                         CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "AKAZE");
img.release();  result.release();
```

10.3.2. Класс *DMatch*

Класс `DMatch` описывает результат сравнения дескрипторов. Инструкция импорта в версии 3.3:

```
import org.opencv.core.DMatch;
```

Инструкция импорта в версии 2.4:

```
import org.opencv.features2d.DMatch;
```

Конструкторы класса:

```
DMatch()
DMatch(int queryIdx, int trainIdx, float distance)
DMatch(int queryIdx, int trainIdx, int imgIdx, float distance)
```

Свойства класса:

```
public int queryIdx
public int trainIdx
public int imgIdx
public float distance
```

Из этих свойств нас чаще всего будет интересовать свойство `distance`, которое описывает расстояние между дескрипторами. Свойство `queryIdx` описывает индекс ключевой точки в исходном изображении, свойство `trainIdx` — индекс найденной ключевой точки в изображении, в котором выполнялся поиск, а свойство `imgIdx` — индекс изображения, в котором найдено совпадение.

Пример:

```
DMatch m = new DMatch();
System.out.println(m);
// DMatch [queryIdx=-1, trainIdx=-1, imgIdx=-1, distance=3.4028235E38]
m.queryIdx = 1;
m.trainIdx = 1;
m.imgIdx = 2;
m.distance = 1.5f;
System.out.println(m);
// DMatch [queryIdx=1, trainIdx=1, imgIdx=2, distance=1.5]
DMatch m2 = new DMatch(1, 1, 1.5f);
System.out.println(m2);
// DMatch [queryIdx=1, trainIdx=1, imgIdx=-1, distance=1.5]
```

10.3.3. Класс *MatOfDMatch*

Класс `MatOfDMatch` реализует матрицу с результатами сравнения дескрипторов. Каждый элемент такой матрицы содержит четыре канала. Инструкция импорта:

```
import org.opencv.core.MatOfDMatch;
```

Иерархия наследования:

```
Object — Mat — MatOfDMatch
```

Конструкторы класса:

```
MatOfDMatch()
MatOfDMatch(DMatch... a)
MatOfDMatch(Mat m)
```

Пример:

```
MatOfDMatch m = new MatOfDMatch(new DMatch(1, 2, 3f),
                                new DMatch(4, 5, 6f));
System.out.println(m.size());        // 1x2
System.out.println(m.channels());    // 4
System.out.println(m.type());        // 29
```

```
System.out.println(CvType.CV_32FC4); // 29
System.out.println(m.dump());
/*
[1, 2, -1, 3;
 4, 5, -1, 6]*/
```

Методы класса `MatOfDMatch`:

- `fromArray()` — заполняет матрицу значениями из массива. Формат метода:

  ```
  public void fromArray(DMatch... a)
  ```

  Пример:

  ```
  MatOfDMatch m = new MatOfDMatch();
  m.fromArray(new DMatch(1, 2, 3f), new DMatch(4, 5, 6f));
  System.out.println(m.dump());
  /*
  [1, 2, -1, 3;
   4, 5, -1, 6]*/
  ```

- `fromList()` — заполняет матрицу значениями из списка. Формат метода:

  ```
  public void fromList(List<DMatch> ldm)
  ```

  Пример:

  ```
  ArrayList<DMatch> list = new ArrayList<DMatch>();
  Collections.addAll(list, new DMatch(1, 2, 3f),
                           new DMatch(4, 5, 6f));
  MatOfDMatch m = new MatOfDMatch();
  m.fromList(list);
  System.out.println(m.dump());
  /*
  [1, 2, -1, 3;
   4, 5, -1, 6]*/
  ```

- `toArray()` — возвращает массив со значениями из матрицы. Формат метода:

  ```
  public DMatch[] toArray()
  ```

  Пример:

  ```
  MatOfDMatch m = new MatOfDMatch(new DMatch(1, 2, 3f),
                                  new DMatch(4, 5, 6f));
  DMatch[] arr = m.toArray();
  System.out.println(Arrays.toString(arr));
  /*
  [DMatch [queryIdx=1, trainIdx=2, imgIdx=-1, distance=3.0],
   DMatch [queryIdx=4, trainIdx=5, imgIdx=-1, distance=6.0]]
  */
  ```

- `toList()` — возвращает список со значениями из матрицы. Формат метода:

  ```
  public List<DMatch> toList()
  ```

Пример:

```
MatOfDMatch m = new MatOfDMatch(new DMatch(1, 2, 3f),
                                new DMatch(4, 5, 6f));
List<DMatch> list = m.toList();
System.out.println(list);
/*
[DMatch [queryIdx=1, trainIdx=2, imgIdx=-1, distance=3.0],
 DMatch [queryIdx=4, trainIdx=5, imgIdx=-1, distance=6.0]]
*/
```

Для создания матрицы из списка можно также воспользоваться статическим методом `vector_DMatch_to_Mat()` из класса `Converters`. Формат метода:

```
import org.opencv.utils.Converters;
public static Mat vector_DMatch_to_Mat(List<DMatch> matches)
```

Выполнить обратную операцию позволяет статический метод `Mat_to_vector_DMatch()` из класса `Converters`. Формат метода:

```
public static void Mat_to_vector_DMatch(Mat m, List<DMatch> matches)
```

Пример:

```
ArrayList<DMatch> list = new ArrayList<DMatch>();
Collections.addAll(list, new DMatch(1, 2, 3f),
                         new DMatch(4, 5, 6f));
Mat m = Converters.vector_DMatch_to_Mat(list);
System.out.println(m.dump());
/*
[1, 2, -1, 3;
 4, 5, -1, 6]*/
System.out.println(m.channels()); // 4
System.out.println(m.type());     // 30
System.out.println(m.size());     // 1x2
ArrayList<DMatch> list2 = new ArrayList<DMatch>();
Converters.Mat_to_vector_DMatch(m, list2);
System.out.println(list2);
/*
[DMatch [queryIdx=1, trainIdx=2, imgIdx=-1, distance=3.0],
 DMatch [queryIdx=4, trainIdx=5, imgIdx=-1, distance=6.0]]
*/
```

Обратите внимание на тип возвращаемой матрицы: класс `MatOfDMatch` возвращает матрицу типа `CV_32FC4`, а методы из класса `Converters` работают с матрицами типа `CV_64FC4`:

```
System.out.println(CvType.CV_32FC4); // 29
System.out.println(CvType.CV_64FC4); // 30
```

10.3.4. Отрисовка найденных совпадений

Нарисовать найденные совпадения ключевых точек позволяет статический метод `drawMatches()` из класса `Features2d`. Форматы метода:

```
import org.opencv.features2d.Features2d;
public static void drawMatches(Mat img1, MatOfKeyPoint keypoints1,
                               Mat img2, MatOfKeyPoint keypoints2,
                               MatOfDMatch matches1to2, Mat outImg)
public static void drawMatches(Mat img1, MatOfKeyPoint keypoints1,
                               Mat img2, MatOfKeyPoint keypoints2,
                               MatOfDMatch matches1to2, Mat outImg,
                               Scalar matchColor, Scalar singlePointColor,
                               MatOfByte matchesMask, int flags)
```

В параметрах `img1` и `img2` указываются исходные изображения, а в параметрах `keypoints1` и `keypoints2` — матрицы с ключевыми точками. В параметре `matches1to2` задается матрица с результатами сравнения ключевых точек, а в параметре `outImg` — матрица, в которую будет записан результат операции. В параметре `matchColor` можно задать цвет совпадающих ключевых точек и линий между ними. Если параметр `matchColor` не задан, или указано значение `Scalar.all(-1)`, то совпадения рисуются разными цветами. В параметре `singlePointColor` можно задать цвет единичных ключевых точек. Если параметр `singlePointColor` не задан, или указано значение `Scalar.all(-1)`, то ключевые точки рисуются разными цветами. Параметр `matchesMask` задает маску. В параметре `flags` можно указать следующие флаги (константы из класса `Features2d`):

- `0` (значение по умолчанию) — для матрицы `outImg` вызывается метод `create()`, потом копируются изображения из матриц `img1` и `img2` и на получившемся изображении рисуются ключевые точки и совпадения;
- `DRAW_OVER_OUTIMG` — ключевые точки рисуются на существующем изображении в матрице `outImg` (метод `create()` не вызывается). Формат:

  ```
  public static final int DRAW_OVER_OUTIMG
  ```

- `NOT_DRAW_SINGLE_POINTS` — единичные ключевые точки не рисуются. Формат:

  ```
  public static final int NOT_DRAW_SINGLE_POINTS
  ```

- `DRAW_RICH_KEYPOINTS` — при рисовании будут учитываться размеры и ориентация ключевой точки (рисуется круг, внутри которого прямой линией из центра круга показывается ориентация). Формат:

  ```
  public static final int DRAW_RICH_KEYPOINTS
  ```

Выведем значения констант:

```
System.out.println(Features2d.DRAW_OVER_OUTIMG);       // 1
System.out.println(Features2d.NOT_DRAW_SINGLE_POINTS); // 2
System.out.println(Features2d.DRAW_RICH_KEYPOINTS );   // 4
```

Если сравнение производилось с помощью метода `knnMatch()` или `radiusMatch()`, то нарисовать найденные совпадения ключевых точек можно с помощью статического метода `drawMatches2()` из класса `Features2d`. Форматы метода:

```
public static void drawMatches2(Mat img1, MatOfKeyPoint keypoints1,
       Mat img2, MatOfKeyPoint keypoints2,
       List<MatOfDMatch> matches1to2, Mat outImg)
public static void drawMatches2(Mat img1, MatOfKeyPoint keypoints1,
       Mat img2, MatOfKeyPoint keypoints2, List<MatOfDMatch> matches1to2,
       Mat outImg, Scalar matchColor, Scalar singlePointColor,
       List<MatOfByte> matchesMask, int flags)
```

10.3.5. Класс *DescriptorMatcher*

Класс `DescriptorMatcher` описывает методы сравнения ключевых точек с использованием различных алгоритмов. Инструкция импорта:

```
import org.opencv.features2d.DescriptorMatcher;
```

Создать объект позволяет статический метод `create()`. Форматы метода:

```
public static DescriptorMatcher create(int matcherType)
public static DescriptorMatcher create(
                        String descriptorMatcherType) // Нет в 2.4
```

В параметре `matcherType` можно указать следующие алгоритмы сравнения (статические константы из класса `DescriptorMatcher`):

```
public static final int FLANNBASED
public static final int BRUTEFORCE
public static final int BRUTEFORCE_L1
public static final int BRUTEFORCE_SL2
public static final int BRUTEFORCE_HAMMING
public static final int BRUTEFORCE_HAMMINGLUT
```

Выведем значения констант:

```
System.out.println(DescriptorMatcher.FLANNBASED);            // 1
System.out.println(DescriptorMatcher.BRUTEFORCE);            // 2
System.out.println(DescriptorMatcher.BRUTEFORCE_L1);         // 3
System.out.println(DescriptorMatcher.BRUTEFORCE_SL2);        // 6
System.out.println(DescriptorMatcher.BRUTEFORCE_HAMMING);    // 4
System.out.println(DescriptorMatcher.BRUTEFORCE_HAMMINGLUT); // 5
```

Если дескрипторы вычислялись алгоритмами `SIFT` или `SURF`, то можно использовать алгоритмы сравнения `BRUTEFORCE_L1` и `BRUTEFORCE`, а если алгоритмами `ORB`, `BRIEF` и `BRISK`, то следует использовать алгоритм сравнения `BRUTEFORCE_HAMMING`.

Выполнить сравнение позволяет метод `match()`. Форматы метода:

```
public void match(Mat queryDescriptors, Mat trainDescriptors,
                  MatOfDMatch matches)
```

```
public void match(Mat queryDescriptors, Mat trainDescriptors,
                  MatOfDMatch matches, Mat mask)
public void match(Mat queryDescriptors, MatOfDMatch matches)
public void match(Mat queryDescriptors, MatOfDMatch matches,
                  List<Mat> masks)
```

В параметре `queryDescriptors` указывается матрица с дескрипторами для первого изображения, а в параметре `trainDescriptors` — матрица с дескрипторами для второго. Результат сравнения будет записан в матрицу `matches`. В параметре `mask` можно указать маску (при условии, что алгоритм сравнения поддерживает маски. Проверить это можно с помощью метода `isMaskSupported()`).

Пример сравнения:

```
MatOfDMatch matches = new MatOfDMatch();
DescriptorMatcher dm = DescriptorMatcher.create(
                          DescriptorMatcher.BRUTEFORCE_HAMMING);
dm.match(descriptors_img, descriptors_img2, matches);
```

Третий и четвертый форматы метода `match()` используются при предварительном добавлении нескольких дескрипторов с помощью метода `add()`. Формат метода:

```
public void add(List<Mat> descriptors)
```

Получить список с добавленными дескрипторами позволяет метод `getTrainDescriptors()`. Формат метода:

```
public List<Mat> getTrainDescriptors()
```

С помощью метода `empty()` можно проверить, были добавлены дескрипторы или нет. Формат метода:

```
public boolean empty()
```

Очистить список с дескрипторами позволяет метод `clear()`. Формат метода:

```
public void clear()
```

После добавления дескрипторов в некоторых случаях требуется вызвать метод `train()` для построения необходимых структур для быстрого поиска (например, при использовании алгоритма сравнения `FLANNBASED`). Формат метода:

```
public void train()
```

Пример сравнения с предварительным добавлением дескрипторов:

```
MatOfDMatch matches = new MatOfDMatch();
DescriptorMatcher dm = DescriptorMatcher.create(
                          DescriptorMatcher.BRUTEFORCE_HAMMING);
ArrayList<Mat> dlist = new ArrayList<Mat>();
dlist.add(descriptors_img2);
System.out.println(dm.empty()); // true
System.out.println(dm.getTrainDescriptors()); // []
dm.add(dlist);
```

```
System.out.println(dm.empty()); // false
System.out.println(dm.getTrainDescriptors());
// [Mat [ 265*32*CV_8UC1, isCont=true, isSubmat=false,
// nativeObj=0x1cb306c0, dataAddr=0x1cb510c0 ]]
dm.train();
dm.match(descriptors_img, matches);
```

Метод `knnMatch()` находит `k` лучших совпадений для каждого дескриптора. Форматы метода:

```
public void knnMatch(Mat queryDescriptors, Mat trainDescriptors,
                     List<MatOfDMatch> matches, int k)
public void knnMatch(Mat queryDescriptors, Mat trainDescriptors,
                     List<MatOfDMatch> matches, int k, Mat mask,
                     boolean compactResult)
public void knnMatch(Mat queryDescriptors, List<MatOfDMatch> matches,
                    int k)
public void knnMatch(Mat queryDescriptors, List<MatOfDMatch> matches,
                    int k, List<Mat> masks, boolean compactResult)
```

Пример нахождения двух лучших совпадений для каждого дескриптора и отрисовки результата сравнения:

```
DescriptorMatcher dm = DescriptorMatcher.create(
                         DescriptorMatcher.BRUTEFORCE_HAMMING);
List<MatOfDMatch> matches2 = new ArrayList<MatOfDMatch>();
dm.knnMatch(descriptors_img, descriptors_img2, matches2, 2);
Mat outImg2 = new Mat(img.rows() + img2.rows() + 10,
                      img.cols() + img2.cols() + 10,
                      CvType.CV_8UC3, CvUtils.COLOR_BLACK);
Features2d.drawMatches2(img, kp_img, img2, kp_img2, matches2, outImg2);
CvUtilsFX.showImage(outImg2, "Результат сравнения");
```

Ограничить результат сравнения расстоянием `maxDistance` (например, расстоянием Хэмминга, а не расстоянием в пикселах) позволяет метод `radiusMatch()`. Форматы метода:

```
public void radiusMatch(Mat queryDescriptors, Mat trainDescriptors,
                        List<MatOfDMatch> matches, float maxDistance)
public void radiusMatch(Mat queryDescriptors, Mat trainDescriptors,
                        List<MatOfDMatch> matches, float maxDistance,
                        Mat mask, boolean compactResult)
public void radiusMatch(Mat queryDescriptors, List<MatOfDMatch> matches,
                        float maxDistance)
public void radiusMatch(Mat queryDescriptors, List<MatOfDMatch> matches,
                        float maxDistance, List<Mat> masks,
                        boolean compactResult)
```

Пример использования метода `radiusMatch()`:

```
DescriptorMatcher dm = DescriptorMatcher.create(
                         DescriptorMatcher.BRUTEFORCE_HAMMING);
```

```
List<MatOfDMatch> matches2 = new ArrayList<MatOfDMatch>();
dm.radiusMatch(descriptors_img, descriptors_img2, matches2, 3);
Mat outImg2 = new Mat(img.rows() + img2.rows() + 10,
                      img.cols() + img2.cols() + 10,
                      CvType.CV_8UC3, CvUtils.COLOR_BLACK);
Features2d.drawMatches2(img, kp_img, img2, kp_img2, matches2, outImg2);
CvUtilsFX.showImage(outImg2, "Результат сравнения");
```

Полный пример использования алгоритма сравнения `BRUTEFORCE_HAMMING` совместно с алгоритмом `ORB` приведен в листинге 10.8. Здесь мы сравним исходное изображение с его копией, повернутой на угол 90 градусов, из полного набора совпадений выберем только самые лучшие совпадения и нарисуем результат (рис. 10.7).

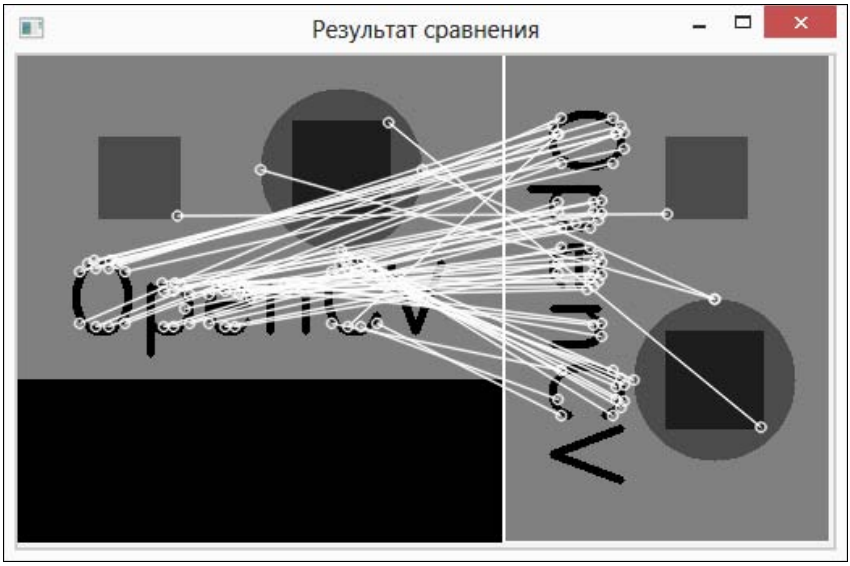


Рис. 10.7. Результат сравнения ключевых точек (листинг 10.8)

Листинг 10.8. Сравнение ключевых точек

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
Mat img2 = Imgcodecs.imread("C:\\book\\opencv\\foto3_90.png");
if (img.empty() || img2.empty()) {
   System.out.println("Не удалось загрузить изображения");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);
Imgproc.cvtColor(img2, img2, Imgproc.COLOR_BGR2GRAY);
// Находим ключевые точки
MatOfKeyPoint kp_img = new MatOfKeyPoint();
MatOfKeyPoint kp_img2 = new MatOfKeyPoint();
FeatureDetector fd = FeatureDetector.create(FeatureDetector.ORB);
```

```
fd.detect(img, kp_img);
fd.detect(img2, kp_img2);
// Отрисовываем найденные ключевые точки
Mat result = new Mat();
Features2d.drawKeypoints(img, kp_img, result,
                         CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "result");
Mat result2 = new Mat();
Features2d.drawKeypoints(img2, kp_img2, result2,
                         CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result2, "result2");
// Вычисляем дескрипторы
DescriptorExtractor extractor = DescriptorExtractor.create(
                                  DescriptorExtractor.ORB);
Mat descriptors_img = new Mat();
Mat descriptors_img2 = new Mat();
extractor.compute(img, kp_img, descriptors_img);
extractor.compute(img2, kp_img2, descriptors_img2);
// Сравниваем дескрипторы
MatOfDMatch matches = new MatOfDMatch();
DescriptorMatcher dm = DescriptorMatcher.create(
                          DescriptorMatcher.BRUTEFORCE_HAMMING);
dm.match(descriptors_img, descriptors_img2, matches);
// Вычисляем минимальное и максимальное значения
double max_dist = Double.MIN_VALUE, min_dist = Double.MAX_VALUE;
float dist = 0;
List<DMatch> list = matches.toList();
for (int i = 0, j = list.size(); i < j; i++) {
   dist = list.get(i).distance;
   if (dist == 0) continue;
   if (dist < min_dist) min_dist = dist;
   if (dist > max_dist) max_dist = dist;
}
System.out.println("min = " + min_dist + " max = " + max_dist);
// Находим лучшие совпадения
LinkedList<DMatch> list_good = new LinkedList<DMatch>();
for (int i = 0, j = list.size(); i < j; i++) {
   if (list.get(i).distance < min_dist * 3) {
      list_good.add(list.get(i));
   }
}
System.out.println(list_good.size());
MatOfDMatch mat_good = new MatOfDMatch();
mat_good.fromList(list_good);
// Отрисовываем результат
Mat outImg = new Mat(img.rows() + img2.rows() + 10,
                     img.cols() + img2.cols() + 10,
                     CvType.CV_8UC3, CvUtils.COLOR_BLACK);
```

```
Features2d.drawMatches(img, kp_img, img2, kp_img2, mat_good, outImg,
      new Scalar(255, 255, 255), Scalar.all(-1), new MatOfByte(),
      Features2d.NOT_DRAW_SINGLE_POINTS);
CvUtilsFX.showImage(outImg, "Результат сравнения");
img.release();              img2.release();
kp_img.release();           kp_img2.release();
descriptors_img.release();  descriptors_img2.release();
matches.release();          mat_good.release();
result.release();           result2.release();
outImg.release();
```

10.4. Создание панорамы

Итак, особые точки мы нашли, а как использовать их на практике? Во-первых, сравнение по особым точкам позволит найти объект с одного изображения на другом (см. листинг 10.8). Причем, этот объект может быть повернут, искажен или иметь другой масштаб. Во-вторых, особые точки используются для стабилизации видео. В-третьих, с помощью особых точек можно совместить два изображения и получить панораму. Существуют и другие области применения.

Давайте попробуем создать панораму из двух фотографий, снятых с одной точки. Для этого потребуется выполнить следующие шаги:

1. Найти ключевые точки на каждом изображении с помощью методов из класса `FeatureDetector`.
2. Вычислить дескрипторы с помощью методов из класса `DescriptorExtractor`.
3. Выполнить сравнение дескрипторов с помощью методов из класса `DescriptorMatcher`.
4. Выбрать самые похожие точки из результатов сравнения.
5. Рассчитать матрицу трансформации и применить ее к одному изображению.
6. Совместить две фотографии в одну.

Первые четыре шага мы уже рассматривали ранее в этой главе. Чтобы рассчитать матрицу трансформации перспективы, нужно воспользоваться статическим методом `findHomography()` из класса `Calib3d`. Форматы метода:

```
import org.opencv.calib3d.Calib3d;
public static Mat findHomography(MatOfPoint2f srcPoints,
                                 MatOfPoint2f dstPoints)
public static Mat findHomography(MatOfPoint2f srcPoints,
                     MatOfPoint2f dstPoints, int method,
                     double ransacReprojThreshold)
public static Mat findHomography(MatOfPoint2f srcPoints,
                     MatOfPoint2f dstPoints, int method,
                     double ransacReprojThreshold, Mat mask) // Нет в 3.3
```

```
public static Mat findHomography(MatOfPoint2f srcPoints,
                    MatOfPoint2f dstPoints, int method,
                    double ransacReprojThreshold, Mat mask,
                    int maxIters, double confidence)        // Нет в 2.4
```

В первом параметре указываются координаты особых точек на одной фотографии, а во втором параметре — координаты соответствующих им точек на другой. Количество точек не должно быть менее четырех. Параметр `method` задает метод вычисления матрицы трансформации. Можно указать значение `0` (значение по умолчанию, используются все точки) или следующие константы из класса `Calib3d`:

```
public static final int RANSAC
public static final int LMEDS
public static final int RHO     // Нет в версии 2.4
```

Выведем значения констант:

```
System.out.println(Calib3d.RANSAC); // 8
System.out.println(Calib3d.LMEDS);  // 4
System.out.println(Calib3d.RHO);    // 16
```

Параметр `ransacReprojThreshold` задает пороговое значение для метода `RANSAC` (значение по умолчанию: `3`), а параметр `mask` — маску. Метод вернет матрицу трансформации, которую можно применить с помощью метода `warpPerspective()` (см. *разд. 5.7*). Если матрицу рассчитать не удалось, то метод вернет пустую матрицу.

Пример создания панорамы приведен в листинге 10.9.

Листинг 10.9. Создание панорамы

```
Mat img_orig = Imgcodecs.imread("C:\\book\\opencv\\panorama1.jpg");
Mat img2_orig = Imgcodecs.imread("C:\\book\\opencv\\panorama2.jpg");
if (img_orig.empty() || img2_orig.empty()) {
   System.out.println("Не удалось загрузить изображения");
   return;
}
Mat img = new Mat();
Mat img2 = new Mat();
Imgproc.cvtColor(img_orig, img, Imgproc.COLOR_BGR2GRAY);
Imgproc.cvtColor(img2_orig, img2, Imgproc.COLOR_BGR2GRAY);
// Находим ключевые точки
MatOfKeyPoint kp_img = new MatOfKeyPoint();
MatOfKeyPoint kp_img2 = new MatOfKeyPoint();
FeatureDetector fd = FeatureDetector.create(FeatureDetector.ORB);
fd.detect(img, kp_img);
fd.detect(img2, kp_img2);
// Вычисляем дескрипторы
DescriptorExtractor extractor = DescriptorExtractor.create(
                                  DescriptorExtractor.ORB);
Mat descriptors_img = new Mat();
Mat descriptors_img2 = new Mat();
```

```
extractor.compute(img, kp_img, descriptors_img);
extractor.compute(img2, kp_img2, descriptors_img2);
// Сравниваем дескрипторы
MatOfDMatch matches = new MatOfDMatch();
DescriptorMatcher dm = DescriptorMatcher.create(
                          DescriptorMatcher.BRUTEFORCE_HAMMING);
dm.match(descriptors_img, descriptors_img2, matches);
// Вычисляем минимальное значение
double min_dist = Double.MAX_VALUE, dist = 0;
List<DMatch> list = matches.toList();
for (int i = 0, j = list.size(); i < j; i++) {
   dist = list.get(i).distance;
   if (dist == 0) continue;
   if (dist < min_dist) min_dist = dist;
}
// Находим лучшие совпадения
LinkedList<DMatch> list_good = new LinkedList<DMatch>();
for (int i = 0, j = list.size(); i < j; i++) {
   if (list.get(i).distance < min_dist * 3) {
      list_good.add(list.get(i));
   }
}
if (list_good.size() < 4) {
   System.out.println("Мало хороших совпадений: " + list_good.size());
   return;
}
MatOfDMatch mat_good = new MatOfDMatch();
mat_good.fromList(list_good);
// Отрисовываем результат сравнения
Mat outImg = new Mat(img.rows() + img2.rows() + 10,
                     img.cols() + img2.cols() + 10,
                     CvType.CV_8UC3, CvUtils.COLOR_WHITE);
Features2d.drawMatches(img, kp_img, img2, kp_img2, mat_good, outImg,
      new Scalar(255, 255, 255), Scalar.all(-1), new MatOfByte(),
      Features2d.NOT_DRAW_SINGLE_POINTS);
CvUtilsFX.showImage(outImg, "Результат сравнения");

// Выбираем 50 лучших точек
list_good.sort(new Comparator<DMatch>() {
   @Override
   public int compare(DMatch obj1, DMatch obj2) {
      if (obj1.distance < obj2.distance) return -1;
      if (obj1.distance > obj2.distance) return 1;
      return 0;
   }
});
```

```
List<KeyPoint> keys1 = kp_img.toList();
List<KeyPoint> keys2 = kp_img2.toList();
LinkedList<Point> list1 = new LinkedList<Point>();
LinkedList<Point> list2 = new LinkedList<Point>();
DMatch dmatch = null;
int count = 50;
if (list_good.size() < count) {
   count = list_good.size();
}
for (int i = 0; i < count; i++) {
   dmatch = list_good.get(i);
   list1.add(keys1.get(dmatch.queryIdx).pt);
   list2.add(keys2.get(dmatch.trainIdx).pt);
}
MatOfPoint2f p1 = new MatOfPoint2f();
MatOfPoint2f p2 = new MatOfPoint2f();
p1.fromList(list1);
p2.fromList(list2);
// Вычисляем матрицу трансформации
Mat h = Calib3d.findHomography(p2, p1, Calib3d.RANSAC, 3);
if (h.empty()) {
   System.out.println("Не удалось рассчитать матрицу трансформации");
   return;
}
// Применяем матрицу трансформации
Mat panorama = new Mat();
Imgproc.warpPerspective(img2_orig, panorama, h,
        new Size(img_orig.width() + img2_orig.width(),
                 img_orig.height() + img2_orig.height()),
        Imgproc.INTER_LINEAR, Core.BORDER_CONSTANT,
        Scalar.all(0));
CvUtilsFX.showImage(panorama, "Результат трансформации");
Mat subMat = panorama.submat(
                new Rect(0, 0, img_orig.width(), img_orig.height()));
img_orig.copyTo(subMat);
CvUtilsFX.showImage(panorama, "Панорама");
img.release();              img2.release();
img_orig.release();         img2_orig.release();
kp_img.release();           kp_img2.release();
descriptors_img.release();  descriptors_img2.release();
matches.release();          mat_good.release();
subMat.release();           panorama.release();
outImg.release();           h.release();
p1.release();               p2.release();
```

Этот способ создания панорамы далеко не идеален. Во-первых, вторая фотография должна быть либо снизу относительно первой, либо справа. При других положениях результат будет неправильным. Во-вторых, никакой проверки правильности сравнения ключевых точек нет. Явные несовпадения метод RANSAC отбросит по пороговому значению, но и это не дает никакой гарантии. В некоторых случаях матрица трансформации создается такая, что после ее применения фотографию не узнать, а уж сопоставить с первой вообще нет шансов. В-третьих, даже после правильной трансформации теряется детализация фотографии, вследствие чего она может очень сильно отличаться от первой фотографии. В-четвертых, очень много зависит от объектов на фотографии и метода поиска ключевых точек: один метод подходит для одних объектов, второй — для других, третий — для третьих, а вот универсального метода нет.

Существует множество и других проблем, связанных, например, с параметрами фотоаппарата при съемке, искажениями объектива, перемещением объектов в кадре и др. Ряд проблем можно решить использованием маски при наложении фотографий, ну а неправильные параметры экспозиции при съемке часто исправить очень сложно. Помните, что снимать панорамы нужно только в ручном режиме (ISO, выдержка, диафрагма и баланс белого должны быть одинаковыми для всех кадров панорамы). Желательно пользоваться штативом, причем со специальными приспособлениями для съемки панорам, т. к. точка поворота должна совпадать с фокальной точкой (а не с точкой штативного гнезда), которая обычно находится внутри объектива. Только в этом случае можно избежать перемещения объектов, расположенных на переднем плане.

10.5. Класс *Feature2D*

Статический метод `create()` из классов `FeatureDetector` и `DescriptorExtractor` создает объекты с настройками по умолчанию. Если нужно указать пользовательские настройки, то в версии 3.3 для поиска ключевых точек можно воспользоваться классами `ORB`, `AKAZE`, `GFTTDetector` (типы `GFTT` и `HARRIS`), `FastFeatureDetector`, `BRISK` и `MSER`, а для вычисления дескрипторов — классами `ORB`, `AKAZE` и `BRISK`. Все классы находятся в пакете `org.opencv.features2d`.

Во всех этих классах есть статический метод `create()`, который позволяет создать объект как с настройками по умолчанию, так и с пользовательскими настройками. Например, в классе `ORB` метод `create()` имеет следующие форматы:

```
import org.opencv.features2d.ORB;
public static ORB create()
public static ORB create(int nfeatures, float scaleFactor, int nlevels,
                         int edgeThreshold, int firstLevel, int WTA_K,
                         int scoreType, int patchSize, int fastThreshold)
```

Пример создания объекта класса `ORB`, поиска ключевых точек и их отрисовки приведен в листинге 10.10.

Листинг 10.10. Класс ORB

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);

ORB orb = ORB.create();
MatOfKeyPoint kp = new MatOfKeyPoint();
orb.detect(img, kp);
Mat result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "ORB");
img.release();  result.release();
```

Помимо метода `create()`, классы содержат множество методов для получения или изменения настроек. Например, класс `GFTTDetector` содержит метод `setHarrisDetector()`, с помощью которого можно выбрать либо тип `GFTT`, либо `HARRIS`.

Пример использования класса `GFTTDetector` приведен в листинге 10.11.

Листинг 10.11. Класс GFTTDetector (тип HARRIS)

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\foto3.png");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
Imgproc.cvtColor(img, img, Imgproc.COLOR_BGR2GRAY);

GFTTDetector fd = GFTTDetector.create();
fd.setHarrisDetector(true);
MatOfKeyPoint kp = new MatOfKeyPoint();
fd.detect(img, kp);
Mat result = new Mat();
Features2d.drawKeypoints(img, kp, result, CvUtils.COLOR_WHITE, 0);
CvUtilsFX.showImage(result, "GFTTDetector");
img.release();  result.release();
```

Иерархия наследования для класса `ORB` выглядит следующим образом (иерархия наследования других классов точно такая же):

```
Object — Algorithm — Feature2D — ORB
```

Выполнить поиск ключевых точек позволяет метод `detect()` из класса `Feature2D`. Форматы метода:

```
public void detect(Mat image, MatOfKeyPoint keypoints)
public void detect(Mat image, MatOfKeyPoint keypoints, Mat mask)
```

В параметре `image` указывается исходное изображение, в параметре `keypoints` — матрица, в которую будет записан результат, а в параметре `mask` — маска.

Вычислить дескрипторы позволяет метод `compute()` из класса `Feature2D`. Формат метода:

```
public void compute(Mat image, MatOfKeyPoint keypoints, Mat descriptors)
```

В параметре `image` указывается исходное изображение, в параметре `keypoints` — матрица с ключевыми точками, а в параметре `descriptors` — матрица, в которую будет записан результат.

Пример поиска ключевых точек и вычисления дескрипторов с помощью класса `ORB`:

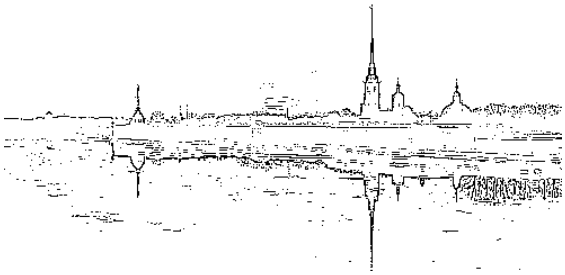
```
ORB orb = ORB.create();
MatOfKeyPoint kp = new MatOfKeyPoint();
orb.detect(img, kp);
Mat descriptors = new Mat();
orb.compute(img, kp, descriptors);
System.out.println(descriptors.size());      // 32x262
System.out.println(descriptors.channels());  // 1
```

С помощью метода `detectAndCompute()` из класса `Feature2D` можно найти ключевые точки и сразу вычислить дескрипторы. Форматы метода:

```
public void detectAndCompute(Mat image, Mat mask,
                             MatOfKeyPoint keypoints, Mat descriptors)
public void detectAndCompute(Mat image, Mat mask,
                             MatOfKeyPoint keypoints, Mat descriptors,
                             boolean useProvidedKeypoints)
```

Пример:

```
ORB orb = ORB.create();
MatOfKeyPoint kp = new MatOfKeyPoint();
Mat descriptors = new Mat();
orb.detectAndCompute(img, new Mat(), kp, descriptors);
System.out.println(kp.size());               // 1x262
System.out.println(descriptors.size());      // 32x262
```



ГЛАВА 11

Каскады Хаара

Если вы хоть раз фотографировали людей на современный фотоаппарат с возможностью предварительного просмотра снимаемого изображения на его дисплее, то наверняка замечали, что при нажатии кнопки спуска на половину ее хода лица людей обводятся рамкой. Таким образом, пока вы нажимали кнопку, автоматика камеры успевала найти и распознать лица. Благодаря этому, наведение фокуса выполняется на лицо, а не на контрастный объект, расположенный на заднем плане. Согласитесь, что лицо человека — более важный объект на фотографии, чем ковер за его спиной.

Поиск лиц основан на *методе Виолы-Джонса*, который был разработан еще в 2001 году. В основе метода лежит интегральное представление изображения *по признакам Хаара*, построение классификатора на основе алгоритма *адаптивного бустинга* и комбинирование классификаторов в каскадную структуру. Алгоритм при поиске работает очень быстро и может быть использован для обнаружения лиц и других объектов в режиме реального времени. Однако у него есть и недостатки — при наклоне лица на угол более 30 градусов вероятность обнаружения лица резко падает.

11.1. Класс *CascadeClassifier*

Выполнить поиск объекта по методу Виолы-Джонса позволяет класс `CascadeClassifier`. Для работы класса требуется классификатор в формате XML, обученный распознавать какие-либо объекты, — например, лица, глаза и т. п. Найти уже готовые к использованию классификаторы можно в папке sources\data\haarcascades, расположенной в каталоге с установленным дистрибутивом OpenCV.

Инструкция импорта:

```
import org.opencv.objdetect.CascadeClassifier;
```

Конструкторы класса:

```
CascadeClassifier()
CascadeClassifier(String filename)
```

Первый конструктор создает объект с настройками по умолчанию. Загрузить классификатор после создания объекта позволяет метод `load()`. Формат метода:

```
public boolean load(String filename)
```

Метод возвращает значение `true`, если классификатор успешно загружен, и `false` — если классификатор загрузить не удалось. Пример проверки загрузки:

```
CascadeClassifier face_detector = new CascadeClassifier();
String path = "C:\\opencv_3_3\\sources\\data\\haarcascades\\";
String name = "haarcascade_frontalface_alt.xml";
if (!face_detector.load(path + name)) {
   System.out.println("Не удалось загрузить классификатор " + name);
   return;
}
```

Второй конструктор позволяет сразу загрузить классификатор из XML-файла. Проверить статус загрузки в этом случае можно с помощью метода `empty()`. Формат метода:

```
public boolean empty()
```

Метод возвращает значение `false`, если классификатор успешно загружен, и `true` — если классификатор загрузить не удалось. Пример проверки загрузки:

```
String path = "C:\\opencv_3_3\\sources\\data\\haarcascades\\";
String name = "haarcascade_frontalface_alt.xml";
CascadeClassifier face_detector = new CascadeClassifier(path + name);
if (face_detector.empty()) {
   System.out.println("Не удалось загрузить классификатор " + name);
   return;
}
```

Поиск объектов выполняется с помощью метода `detectMultiScale()`. Форматы метода:

```
public void detectMultiScale(Mat image, MatOfRect objects)
public void detectMultiScale(Mat image, MatOfRect objects,
               double scaleFactor, int minNeighbors, int flags,
               Size minSize, Size maxSize)
```

В первом параметре указывается исходное изображение (тип `CV_8U`), а во втором — матрица, в которую будет записан результат поиска. Параметр `scaleFactor` задает коэффициент масштабирования (поиск объектов производится на разных масштабах изображения). Значение по умолчанию: `1.1`. В параметре `minNeighbors` можно указать размер соседства (значение по умолчанию: `3`). Параметр `flags` не используется (значение по умолчанию: `0`). Минимальный размер прямоугольника можно задать в параметре `minSize`, а максимальный — в параметре `maxSize`.

11.2. Поиск лиц

Выполнить поиск лиц на фотографии позволяют следующие классификаторы:

- `haarcascade_frontalface_alt.xml`;
- `haarcascade_frontalface_alt_tree.xml`;
- `haarcascade_frontalface_alt2.xml`;
- `haarcascade_frontalface_default.xml`;
- `haarcascade_profileface.xml`.

Пример поиска лиц на фотографии приведен в листинге 11.1, а результат выполнения кода из листинга 11.1 показан на рис. 11.1.

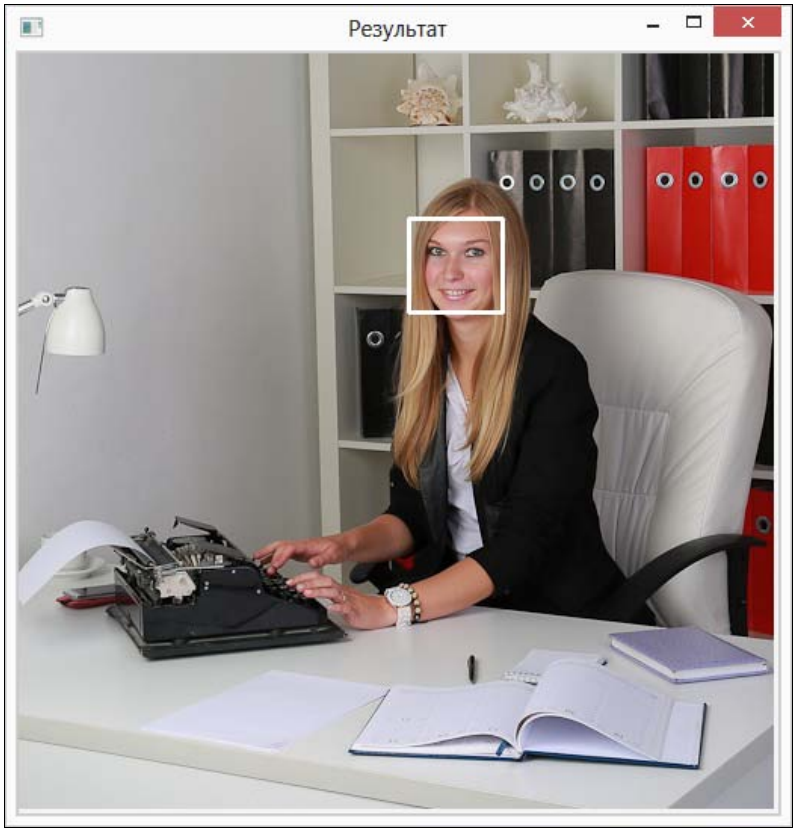


Рис. 11.1. Результат поиска лица (листинг 11.1)

Листинг 11.1. Поиск лиц на фотографии

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\luba1.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
```

```
   return;
}
CascadeClassifier face_detector = new CascadeClassifier();
String path = "C:\\opencv_3_3\\sources\\data\\haarcascades\\";
String name = "haarcascade_frontalface_alt.xml";
if (!face_detector.load(path + name)) {
   System.out.println("Не удалось загрузить классификатор " + name);
   return;
}

MatOfRect faces = new MatOfRect();
face_detector.detectMultiScale(img, faces);
for (Rect r : faces.toList()) {
   Imgproc.rectangle(img, new Point(r.x, r.y),
         new Point(r.x + r.width, r.y + r.height),
         CvUtils.COLOR_WHITE, 2);
}
CvUtilsFX.showImage(img, "Результат");
img.release();
```

11.3. Поиск глаз

Выполнить поиск глаз позволяет классификатор `haarcascade_eye.xml`. Чтобы было меньше ложных срабатываний, лучше вначале найти лицо, а затем уже внутри найденного прямоугольника выполнять поиск глаз.

Пример поиска глаз приведен в листинге 11.2, а результат выполнения кода из листинга 11.2 показан на рис. 11.2.

Листинг 11.2. Поиск глаз

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\luba2.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CascadeClassifier face_detector = new CascadeClassifier();
String path = "C:\\opencv_3_3\\sources\\data\\haarcascades\\";
String name = "haarcascade_frontalface_alt.xml";
if (!face_detector.load(path + name)) {
   System.out.println("Не удалось загрузить классификатор " + name);
   return;
}
CascadeClassifier eye_detector = new CascadeClassifier();
name = "haarcascade_eye.xml";
if (!eye_detector.load(path + name)) {
   System.out.println("Не удалось загрузить классификатор " + name);
```

```
   return;
}
MatOfRect faces = new MatOfRect();
face_detector.detectMultiScale(img, faces);
for (Rect r : faces.toList()) {
   Mat face = img.submat(r);
   MatOfRect eyes = new MatOfRect();
   eye_detector.detectMultiScale(face, eyes);
   for (Rect r2 : eyes.toList()) {
      Imgproc.rectangle(face, new Point(r2.x, r2.y),
            new Point(r2.x + r2.width, r2.y + r2.height),
            CvUtils.COLOR_WHITE, 1);
   }
}
CvUtilsFX.showImage(img, "Результат");
img.release();
```

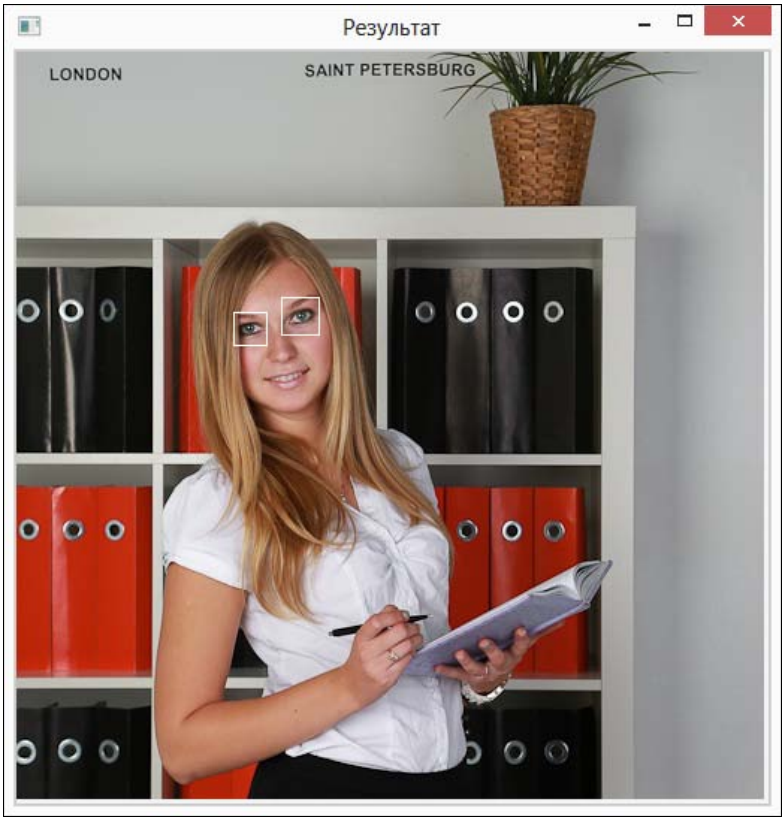


Рис. 11.2. Результат поиска глаз (листинг 11.2)

11.4. Поиск улыбки

Определить, улыбается человек на фотографии или нет, позволяет классификатор `haarcascade_smile.xml`.

Пример поиска улыбки приведен в листинге 11.3, а результат выполнения кода из листинга 11.3 показан на рис. 11.3.

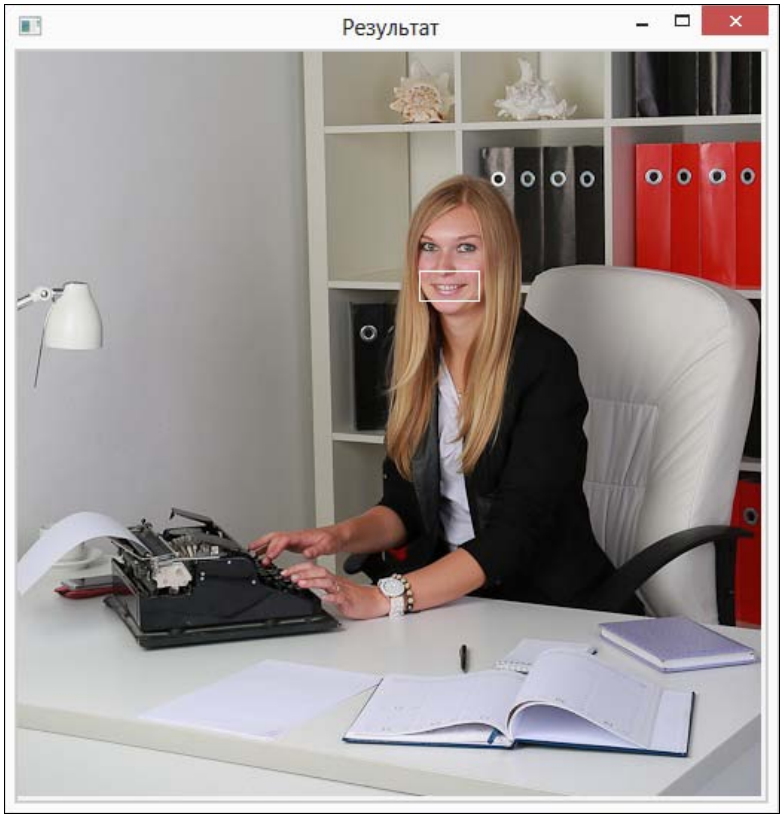


Рис. 11.3. Результат поиска улыбки (листинг 11.3)

Листинг 11.3. Поиск улыбки

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\luba1.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CascadeClassifier face_detector = new CascadeClassifier();
String path = "C:\\opencv_3_3\\sources\\data\\haarcascades\\";
String name = "haarcascade_frontalface_alt.xml";
if (!face_detector.load(path + name)) {
   System.out.println("Не удалось загрузить классификатор " + name);
```

```
   return;
}
CascadeClassifier smile_detector = new CascadeClassifier();
name = "haarcascade_smile.xml";
if (!smile_detector.load(path + name)) {
   System.out.println("Не удалось загрузить классификатор " + name);
   return;
}
MatOfRect faces = new MatOfRect();
face_detector.detectMultiScale(img, faces);
for (Rect r : faces.toList()) {
   Mat face = img.submat(r);
   MatOfRect smile = new MatOfRect();
   smile_detector.detectMultiScale(face, smile);
   for (Rect r3 : smile.toList()) {
      Imgproc.rectangle(face, new Point(r3.x, r3.y),
            new Point(r3.x + r3.width, r3.y + r3.height),
            CvUtils.COLOR_WHITE, 1);
   }
}
CvUtilsFX.showImage(img, "Результат");
img.release();
```

11.5. Поиск носа

Для поиска носа можно воспользоваться классификатором `haarcascade_mcs_nose.xml` (классификатор доступен только в составе дистрибутива версии 2.4).

Пример поиска носа приведен в листинге 11.4, а результат выполнения кода из листинга 11.4 показан на рис. 11.4.

Листинг 11.4. Поиск носа

```
Mat img = Imgcodecs.imread("C:\\book\\opencv\\luba2.jpg");
if (img.empty()) {
   System.out.println("Не удалось загрузить изображение");
   return;
}
CascadeClassifier face_detector = new CascadeClassifier();
String path = "C:\\opencv_2_4\\sources\\data\\haarcascades\\";
String name = "haarcascade_frontalface_alt.xml";
if (!face_detector.load(path + name)) {
   System.out.println("Не удалось загрузить классификатор " + name);
   return;
}
CascadeClassifier nose_detector = new CascadeClassifier();
name = "haarcascade_mcs_nose.xml";
```

```
if (!nose_detector.load(path + name)) {
   System.out.println("Не удалось загрузить классификатор " + name);
   return;
}
MatOfRect faces = new MatOfRect();
face_detector.detectMultiScale(img, faces);
for (Rect r : faces.toList()) {
   Mat face = img.submat(r);
   MatOfRect nose = new MatOfRect();
   nose_detector.detectMultiScale(face, nose);
   for (Rect r3 : nose.toList()) {
      Imgproc.rectangle(face, new Point(r3.x, r3.y),
            new Point(r3.x + r3.width, r3.y + r3.height),
            CvUtils.COLOR_WHITE, 1);
   }
}
CvUtilsFX.showImage(img, "Результат");
img.release();
```

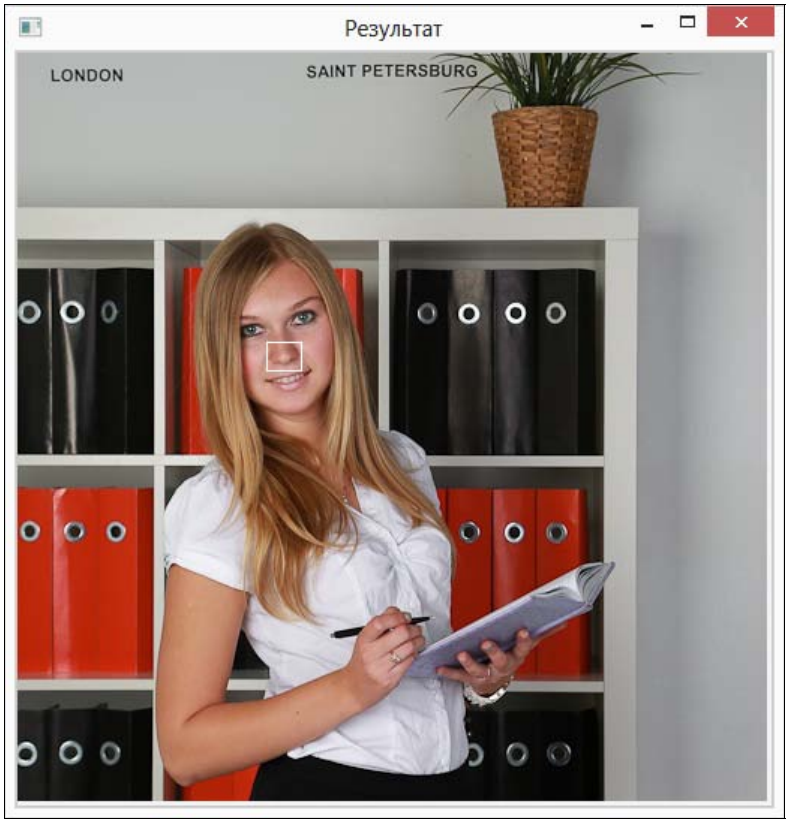


Рис. 11.4. Результат поиска носа (листинг 11.4)

Помимо рассмотренных классификаторов существует множество других, обученных искать пешеходов, номерные знаки на автомобилях, дорожные знаки и т. д. При большом желании вы можете обучить свой собственный классификатор, но для этого потребуется много времени и сил. Во-первых, нужно будет подобрать несколько сот и даже тысяч положительных примеров и не меньшее число отрицательных. Во-вторых, обучение выполняется очень долго и может занять весь день. В-третьих, при недостаточном количестве примеров может вообще ничего не получиться в результате. Зато, если получится, положительные эмоции вам гарантированы.

Обучение каскада Хаара производится с помощью консольных программ opencv_createsamples.exe и opencv_traincascade.exe, расположенных в папке C:\opencv_3_3\build\x64\vc14\bin. В версии 2.4 есть также программа opencv_haartraining.exe. Описание работы с этими программами смотрите в документации.

Если вас заинтересовала возможность обучения, то следует также обратить внимание на два пакета: `org.opencv.ml` и `org.opencv.dnn`. Пакет `org.opencv.ml` содержит классы для машинного обучения, а пакет `org.opencv.dnn` — классы, предназначенные для загрузки моделей глубоких сверточных нейронных сетей, обученных в библиотеках Caffe (**http://caffe.berkeleyvision.org/**), TensorFlow (**https://www.tensorflow.org/**) и Torch (**http://torch.ch/**).

Работа с нейронными сетями потребует от вас не только создания большого объема обучающей выборки, но и хорошего знания математики. Следует также учитывать, что обучение нейронной сети в библиотеках Caffe и TensorFlow производится на языке программирования Python, а в библиотеке Torch — на языке Lua. Однако существует возможность использовать для обучения язык Java. Для библиотеки Caffe есть неофициальная обертка jCaffe (**https://github.com/fastturtle/jCaffe**), а для TensorFlow — экспериментальная версия под Java (без гарантий, но зато от официального разработчика: **https://www.tensorflow.org/install/install_java**).

Заключение

Вот и закончилось наше путешествие в мир OpenCV. Материал книги описывает лишь основы этой замечательной технологии. А здесь мы уточним, где найти дополнительную информацию, чтобы продолжить изучение.

Самым важным источником информации является официальный сайт библиотеки OpenCV: **http://opencv.org/**. На этом сайте вы найдете дистрибутивы, новости, а также ссылки на все другие ресурсы в Интернете, посвященные OpenCV.

На сайте **http://docs.opencv.org/** расположена документация по OpenCV, которая обновляется в режиме реального времени. Библиотека постоянно совершенствуется, появляются новые классы и методы, изменяются параметры методов, добавляются пакеты и т. д. Не забывайте почаще сюда заходить.

На сайте **http://answers.opencv.org/** находится официальный форум по OpenCV. Вы можете задать свой собственный вопрос, просто изучить проблемы других пользователей или помочь другому пользователю в решении его проблемы.

На сайте **http://opencv-java-tutorials.readthedocs.io/en/latest/** можно найти официальный учебник по OpenCV применительно к языку программирования Java.

В состав дистрибутива входят примеры использования библиотеки OpenCV на различных языках программирования. Для языка Java очень мало примеров, а вот для языков C++ и Python примеров достаточно много, и большинство из них весьма интересные. Обязательно рассмотрите эти примеры.

Если в процессе изучения OpenCV у вас возникнет какое-либо недопонимание, то не следует забывать, что Интернет предоставляет множество ответов на самые разнообразные вопросы, — достаточно в строке запроса поискового портала (например, **http://www.google.com/**) набрать свой вопрос. Наверняка уже кто-то сталкивался с подобной проблемой и описал ее решение на каком-либо сайте.

Свои замечания и пожелания вы можете оставить на странице книги на сайте издательства «БХВ-Петербург»: **http://www.bhv.ru/**. А сообщения обо всех замеченных опечатках и неточностях прошу присылать на E-mail: **mail@bhv.ru** — не забудьте только указать название книги и имя автора.

ПРИЛОЖЕНИЕ

Описание электронного архива

Электронный архив к книге выложен на FTP-сервер издательства «БХВ-Петербург» по адресу: **ftp://ftp.bhv.ru/9785977539555.zip**. Ссылка на архив доступна и со страницы книги на сайте **www.bhv.ru**.

Структура архива представлена в табл. П.1.

Таблица П.1. *Структура электронного архива*

Файл или папка	**Описание**
Папка Images	Содержит все изображения, используемые в листингах
Папка Pictures	Содержит все иллюстрации из книги
Listings.doc	Содержит все пронумерованные листинги из книги
Readme.txt	Описание электронного архива

Предметный указатель

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Н

О

П

Р

С

Т

У

Ф

Х

Ц

Ч

Ш

Щ

Э

Я