Содержание

Предисловие

Эта книга представляет собой руководство по библиотеке компьютерного зрения на C++ с открытым исходным кодом (OpenCV) версии 3.x и содержит общие сведения о дисциплине компьютерного зрения в объеме, достаточном для эффективной работы с OpenCV.

Назначение книги

Своим быстрым развитием компьютерное зрение обязано четырем причинам:

- в наш быт вошли мобильные телефоны, а вместе с ними в руках миллионов людей оказались фото- и видеокамеры;
- в Интернете и в поисковых системах накопились гигантские объемы изображений и видео, хранящиеся в огромных базах данных;
- компьютеры, обладающие большой вычислительной мощностью, стали доступным потребительским товаром;
- сами алгоритмы машинного зрения стали более зрелыми (теперь уже и благодаря внедрению глубоких нейронных сетей, поддержка которых в OpenCV неуклонно расширяется; см. модуль *dnn* на странице https://github.com/opencv/opencv_contrib [opencv_contrib]).

OpenCV сыграла важную роль в развитии компьютерного зрения, дав возможность сотням тысяч людей работать более продуктивно. В наши дни OpenCV 3.x позволяет студентам, исследователям, профессионалам и любителям эффективно реализовывать новые проекты, предоставляя внутренне согласованную архитектуру компьютерного зрения в виде библиотеки на языке C++, оптимизированной для различных платформ.

Авторы ставили перед собой следующие задачи:

- полностью документировать OpenCV, точно описав, что означают параметры каждой функции и как эти функции правильно использовать;
- дать читателю интуитивное понимание алгоритмов машинного зрения;
- объяснить читателю, какие алгоритмы в каких случаях следует использовать;
- облегчить читателю реализацию алгоритмов компьютерного зрения и машинного обучения, предложив работающие примеры, от которых можно оттолкнуться;
- подсказать, что делать в сложных случаях, когда что-то не получается.

Эта книга устроена так, чтобы читатель поскорее перешел к интересным и доставляющим удовольствие вещам. Нашими интуитивными объяснениями принципов работы алгоритмов читатель сможет руководствоваться при проектировании и отладке приложений компьютерного зрения. Заодно такой подход делает формальные описания алгоритмов компьютерного зрения и машинного обучения более простыми для усвоения и запоминания.

На кого рассчитана эта книга

Эта книга содержит описания, примеры кода и объяснения различных инструментов, включенных в библиотеку OpenCV 3.x, написанную на C++. Поэтому она может оказаться полезной для различных категорий пользователей.

Профессиональные разработчики

Для профессионального разработчика, которому нужно быстро разработать прототип или реализовать полноценную систему компьютерного зрения, пример кода послужит отправной точкой. Благодаря описаниям алгоритмов читатель узнает или вспомнит, как они работают. Библиотека OpenCV 3.x располагается поверх *уровня аппаратного ускорения* (HAL), поэтому реализованные алгоритмы работают эффективно, в полной мере используя возможности конкретной аппаратной платформы.

Студенты

Такого учебника нам не хватало, когда мы учились. Интуитивно понятные объяснения, детальная документация и примеры кода – все это поможет быстрее продвигаться в изучении компьютерного зрения, работать над более интересными групповыми проектами и в конечном итоге внести свой вклад в отрасль.

Преподаватели

Компьютерное зрение – быстро развивающаяся область. Но опыт показал, что обучение проходит эффективнее, если студенты быстро осваивают материал по доступному учебнику, а преподаватель дает формальные пояснения там, где это необходимо, дополняя их ссылками на современные журнальные статьи и лекциями приглашенных экспертов. Это позволяет студентам быстрее перейти к групповым проектам и заняться более амбициозными задачами.

Любители

Компьютерное зрение – это интересно. А в книге описано, как к нему подступиться.

Нашей основной целью было снабдить читателя достаточным количеством интуитивно понятных пояснений, документации и работающего кода, чтобы он мог поскорее приступить к реализации приложений компьютерного зрения, работающих в режиме реального времени.

На кого эта книга не рассчитана

Это не формальный учебник. В некоторых местах мы приводим математические детали[1], но только для того, чтобы лучше развить интуицию, необходимую для понимания алгоритмов, или пояснить, на что влияют встроенные в алгоритмы предположения. Мы не пытались давать формальные математические доказательства и тем рискуем навлечь на себя гнев авторов, предпочитающих строгий стиль изложения.

Эта книга больше прикладного характера. Она, безусловно, поможет овладеть предметом в целом, но не ориентирована на узкоспециализированные дисциплины (например, обработку медицинских изображений или анализ данных дистанционного зондирования).

Вместе с тем мы полагаем, что, познакомившись с нашими объяснениями, студенты будут не только лучше понимать теорию, но и надолго запомнят ее. Поэтому книга станет хорошим дополнением к теоретическому курсу, а также отличным пособием для вводного курса или для работы над конкретным проектом.

---

[1] Но всегда предупреждаем менее склонных к математике читателей, что они вольны пропустить такие разделы.

О примерах программ

Все приведенные в книге примеры основаны на версии OpenCV 3.x. Код должен работать в операционных системах Linux, Windows и OS X. OpenCV 3.x также в полной мере поддерживает платформы Android и iOS. Исходный код примеров можно скачать с сайта книги по адресу http://shop.oreilly.com/product/0636920044765.do, исходный код OpenCV доступен на GitHub (https://github.com/opencv/opencv), а откомпилированный код для разных платформ – на сайте SourceForge (http://sourceforge.net/projects/opencvlibrary).

OpenCV продолжает активно разрабатываться, официальные версии выходят раз в квартал. Чтобы всегда находиться на острие разработки, скачивайте обновления прямо с сайта на GitHub. OpenCV посвящен сайт http://opencv.org, а для разработчиков имеется вики по адресу https://github.com/opencv/opencv/wiki.

Требования к читателю

От читателя требуется в основном умение программировать на C++. Многие насыщенные математикой разделы факультативны, и об этом имеются предупреждения. Из математики нужно знать основы линейной и матричной алгебры, а также иметь представление о решении задач оптимизации методом наименьших квадратов и базовые знания о нормальном распределении, правиле Байеса и производных простых функций.

Вся математика в этой книге лишь подкрепляет развитие интуиции. Читатель может пропустить математические выкладки и описания алгоритмов, а использовать только определения функций и примеры кода, чтобы откомпилировать приложение и наблюдать за его работой.

Рекомендации по работе с книгой

Книгу не обязательно читать последовательно. Можно рассматривать ее как своего рода руководство пользователя: найти нужную функцию и прочитать описание, если интересно понять принцип ее работы. Но истинное назначение книги – служить учебным пособием. Читатель почерпнет из нее понимание основ компьютерного зрения, а также подробные сведения о том, как и когда применять конкретные алгоритмы.

Книга написана так, чтобы ее можно было использовать в качестве дополнительного или основного учебника по курсу компьютерного зрения для студентов средних и старших курсов. Предполагается, что студенты прочитают ее, чтобы бегло познакомиться с тематикой, а затем дополнят полученные знания материалом из более формальных учебников и из научных статей. В конце каждой главы имеются упражнения для проверки усвоения прочитанного и выработки дополнительных интуитивных представлений.

Подходить к чтению книги можно по-разному.

Сборная солянка

В первом заходе прочитайте главы 1–5, а затем переходите к тем главам или разделам, которые нужны в данный момент. Книгу не обязательно читать последовательно, за исключением глав 18 и 19 (где речь идет о калибровке камеры и получении стереоизображений) и глав 20, 21 и 22 (о машинном обучении). Такой способ больше под-

ходит профессионалам, занятым разработкой конкретного приложения, и студентам, работающим над курсовым проектом.

Систематическое изучение

Читайте по две главы в неделю, так вы проработаете главы 1–22 за 11 недель (на главу 23 много времени не уйдет). Работайте над проектами и углубляйтесь в детали выбранной области, привлекая дополнительные учебники и статьи.

Спринт

Пролистайте главы 1–23 в темпе, соответствующем вашему уровню знаний. Затем переходите к конкретным проектам и углубляйтесь в детали выбранной области, привлекая дополнительные книги и статьи. Такой вариант подходит прежде всего, профессионалам, но может пригодиться и при прохождении более продвинутого курса компьютерного зрения.

Короткая глава 20 содержит общие основы машинного обучения, а в главах 21 и 22 приводятся детали алгоритмов машинного обучения, реализованных в OpenCV, и рекомендации по их применению. Конечно, машинное обучение – неотъемлемая часть распознавания объектов и играет значительную роль в компьютерном зрении, но о нем можно написать отдельную книгу. Специалистам этот текст может послужить отправной точкой для изучения литературы – или хотя бы поможет понять, как подступиться к коду в этой части библиотеки. В версии OpenCV 3.x интерфейс с алгоритмами машинного обучения существенно упрощен и унифицирован.

Мы предпочитаем такой подход к преподаванию компьютерного зрения: быстро пробежаться по содержанию курса, стремясь к тому, чтобы студенты уяснили основные принципы, а затем перейти к интересным групповым проектам, восполняя недостающие формальные детали ссылками на другие учебники и статьи в профильных журналах. Один и тот же метод применим к полусеместровым, семестровым и годичным курсам. Студенты быстро понимают суть поставленной задачи и пишут для нее код. Когда проекты становятся более трудными и требующими больше времени, преподаватель помогает в разработке и отладке сложных систем.

В случае более длительных курсов сам проект может стать источником знаний о методах управления проектами. Сначала строится работающая система, затем она совершенствуется на основе дополнительных знаний, а потом используется для исследования. Результат такого проекта должен заслуживать публикации в материалах какой-то конференции, и предполагается, что будет опубликовано несколько статей по результатам работы уже после окончания курса. Если говорить об OpenCV 3.x, то структура кода на C++, роботы сборки, использование GitHub, анализ запросов на включение кода, автономные и регрессионные тесты, а также документация – все это отличные примеры инфраструктуры профессионально написанного программного обеспечения, которые могут взять за образец стартапы и другие коммерческие организации.

Графические выделения

В книге применяются следующие графические выделения:

Курсив

Новые термины, имена и расширения имен файлов, каталогов и утилит Unix.

`Моноширинный`
Команды, параметры, флаги, имена переменных, атрибутов, ключей, функций, типов, классов, пространств имен, методов, модулей, свойств, а также значения, объекты, события, обработчики событий, XML-теги, HTML-теги, содержимое файлов и результаты работы команд.

`Моноширинный полужирный`
Команды и иные строки, которые следует вводить буквально. Также используется для выделения в примерах кода.

`Моноширинный курсив`
Текст, вместо которого следует подставить значения, заданные пользователем.

[...]
Библиографическая ссылка


Так обозначается совет, рекомендация или замечание общего характера.


Так обозначается предупреждение или предостережение.

О примерах кода

Дополнительные материалы (примеры кода, упражнения и т. д.) размещены по адресу https://github.com/oreillymedia/Learning-OpenCV-3_examples.

Библиотека OpenCV свободна для использования в коммерческих и исследовательских целях, и те же правила применимы к примерам кода в этой книге. Можете использовать их при выполнении домашних заданий, в научно-исследовательских проектах и в коммерческих продуктах! Мы будем весьма признательны, если вы при этом дадите ссылку на книгу, хотя и не настаиваем. В ссылке обычно указываются название книги, имя автора, издательство и ISBN, например: «Learning OpenCV 3 by Adrian Kaehler and Gary Bradski (O'Reilly). Copyright 2017 Adrian Kaehler, Gary Bradski, 978-1-491-93799-0».

Нам интересно знать не только о том, как книга помогла вам сделать домашнюю работу (это как раз лучше хранить в секрете), но и о том, как вы используете OpenCV в академических исследованиях, университетских курсах и в коммерческих программах. Это, конечно, не обязательно, но мы были бы рады получить от вас строчку-другую.

Нам интересно ваше мнение

Вопросы и замечания по поводу этой книги отправляйте в издательство:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США и Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

Для этой книги имеется веб-страница, на которой публикуются примеры и планы на будущее. Адрес страницы: http://bit.ly/learningOpenCV3.

Замечания и вопросы технического характера следует отправлять по адресу bookquestions@oreilly.com.

Дополнительную информацию о наших книгах, курсах, конференциях и новостях вы можете найти на нашем сайте по адресу http://www.oreilly.com.

Читайте нас на Facebook: http://facebook.com/oreilly.

Следите за нашей лентой в Twitter: http://twitter.com/oreillymedia.

Смотрите нас на YouTube: http://www.youtube.com/oreillymedia.

Благодарности

В любом длительном проекте ПО с открытым исходным кодом люди приходят и уходят, оставляя тот или иной вклад. Список всех, кто принимал участие в разработке библиотеки, слишком длинный, и мы не можем его здесь опубликовать, но он имеется в файле *.../opencv/docs/HTML/Contributors/doc_contributors.html*, который входит в состав дистрибутива OpenCV.

Благодарности за помощь в разработке OpenCV

Библиотека родилась в компании Intel, которая заслуживает всяческой благодарности за поддержку проекта на этапах его становления и развития. Время от времени Intel все еще финансирует конкурсы и передает результаты внутренних разработок в OpenCV. Intel также безвозмездно передала код встроенных примитивов производительности, которые прозрачно обеспечивают ускорение на платформах с архитектурой Intel. Спасибо ей за это.

Google стабильно поддерживала разработку OpenCV, финансово поощряя работу молодых специалистов над OpenCV в рамках своего проекта Google Summer of Code; благодаря этому финансированию было проделано немало полезной работы. Компания Willow Garage несколько лет предоставляла средства, позволившие перейти от версии OpenCV 2.x к версии 3.0. В течение этого времени компания Itseez, занимающаяся исследованиями в области компьютерного зрения (недавно ее купила Intel Corporation), обеспечивала инженерную поддержку и предоставляла хостинг для вебслужб. Intel подтвердила устное согласие на продолжение этой поддержки (спасибо!).

Что касается программной стороны дела, то особого упоминания заслуживают несколько лиц, и прежде всего из российской команды. Главным из них является ведущий программист из России Вадим Писаревский, который написал больше библиотечного кода, чем любой другой отдельно взятый человек. Вадим также нянчил библиотеку и управлял ее развитием в тощие годы, когда подъем интереса сменился спадом, дождавшись противоположного тренда. Если и существует настоящий герой в истории библиотеки, то это именно он. Его технические знания также очень помогли при написании этой книги. В плане руководства ему оказывал помощь Виктор Ерухимов, сооснователь Itseez [Itseez], а ныне генеральный директор компании Itseez3D.

В ходе еженедельных встреч нам постоянно помогали управлять разработкой библиотеки Грейс Весом (Grace Vesom), Винсент Рабу (Vincent Rabaud), Стефано Фабри (Stefano Fabri) и, конечно, Вадим Писаревский. Заметки с этих встреч опубликованы по адресу https://github.com/opencv/opencv/wiki/Meeting_notes.

Немало людей внесло вклад в OpenCV в разное время; перечислим лишь недавних авторов: Динар Ахматнуров, Пабло Алькантарилла (Pablo Alcantarilla), Александр Алехин, Даниэл Ангелов (Daniel Angelov), Дмитрий Анисимов, Анатолий Бакшеев, Кристин Балинт (Cristian Balint), Александр Бенуа (Alexandre Benoit), Доран Берже (Laurent Berger), Леонид Бейненсон, Александр Боков, Александр Бовырин, Хилтон Бристоу (Hilton Bristow), Владимит Быстрицкий, Антонелла Каскителли (Antonella Cascitelli), Мануэла Чесса (Manuela Chessa), Эрик Кристиансен (Eric Christiansen), Фредерик Деверне (Frederic Devernay), Мария Димашова, Роман Донченко, Владимир Дудник, Виктор Ерухимов, Георгиос Евагелидис (Georgios Evangelidis), Стефано Фабри (Stefano Fabri), Серджио Гарридо (Sergio Garrido), Гаррис Гаспаракис (Harris Gasparakis), Юрий Гитман, Луис Гомес (Lluis Gomez), Юрий Горбачев, Елена Гвоздева, Филипп Хаспер (Philipp Hasper), Фернандо Х. Иглесиас Гарсиа (Fernando J. Iglesias Garcia), Александр Калистратов, Андрей Камаев, Александр Карсаков, Рауль Кави (Rahul Kavi), Пэт О'Кифи (Pat O'Keefe), Сиддхартх Кхерада (Siddharth Kherada), Евгений Хведченя, Анна Коган, Марина Колпакова, Кирилл Корняков, Иван Королев, Максим Костин, Евгений Кожинов, Илья Крылов, Лакосоно Курнианггоро (Laksono Kurnianggoro), Бай Чжень Лай (Baisheng Lai), Илья Лавренев, Алекс Леонтьев, Джил Ливай (Gil Levi), Бо Ли (Bo Li), Илья Лысенков, Виталий Людвиченко, Бенс Мадьяр (Bence Magyar), Никита Манович, Хуан Мануэль Перес Руа (Juan Manuel Perez Rua), Константин Мацкевич, Патрик Михелич (Patrick Mihelich), Александр Мордвинцев, Федор Морозов, Грегори Морс (Gregory Morse), Мариус Муя (Marius Muja), Мирча Пауль Муресан (Mircea Paul Muresan), Сергей Носов, Даниил Осокин, Сеон-Вук Парк (Seon-Wook Park), Андрей Павленко, Александр Петриков, Philip aka Dikay900, Прасанна (Prasanna), Франческо Пужа (Francesco Puja), Стивен Путтеманс (Steven Puttemans), Винсент Рабу (Vincent Rabaud), Эдгар Риба (Edgar Riba), Соди Ригни (Cody Rigney), Павел Ройтберг, Этан Рабли (Ethan Rublee), Альфонсо Санчес-Беато (Alfonso Sanchez-Beato), Андрей Сенин, Максим Шабунини, Влад Шахуро, Ади Шавит (Adi Shavit), Александр Шишков, Сергей Сиволгин, Марвин Смит (Marvin Smith), Александр Сморкалов, Фабио Солари (Fabio Solari), Адриан Стратулат (Adrian Stratulat), Евгений Таланин, Мануэле Тамбуррано (Manuele Tamburrano), Озан Тонкал (Ozan Tonkal), Владимир Тян, Янник Верди (Yannick Verdie), Пьер-Эммануэль Вьель (Pierre-Emmanuel Viel), Владислав Виноградов, Павел Власов, Филипп Вагнер (Philipp Wagner), Ида Ванг (Yida Wang), Чжао Лонг Сю (Jiaolong Xu), Мариан Зайко (Marian Zajko), Зоран Жижкович (Zoran Zivkovic).

Время от времени в файле https://github.com/opencv/opencv/wiki/ChangeLog появляются и другие имена. Наконец, в настоящее время нам помогает в сопровождении сайта OpenCV.org компания Arraiy [Arraiy].

Благодарности за помощь в работе над книгой

Мы благодарим Джона Маркоффа (John Markoff), научного обозревателя из «Нью-Йорк Таймс», который помогал нам при подготовке этого и предыдущего издания книги моральной поддержкой, важными знакомствами и общими рекомендациями по написанию текстов – плодами многолетней работы «на передовой». Мы также благодарны многочисленным редакторам в издательстве O'Reilly, особенно Дону Шанафельту (Dawn Schanafelt), который терпеливо продолжал работать с нами, когда просрочки стали нормой из-за того, что заблудшие авторы затеяли основывать

стартап. Этот проект растянулся надолго и плавно перешел от OpenCV 2.x к текущей версии OpenCV 3.x. Большое спасибо издательству O'Reilly, не бросившему нас на этом пути.

Дополнения от Адриана...

В первом издании книги я отметил несколько замечательных учителей, которые помогли мне достигнуть уровня, необходимого для такой работы. За прошедшие с тех пор годы ценность уроков, полученных от каждого из них, стала еще более очевидной. Большое спасибо всем им вместе и каждому в отдельности. Хотел бы добавить к этому списку выдающихся наставников Тома Томбрелло (Tom Tombrello), перед которым я в неоплатном долгу и чьей памяти посвящаю свой вклад в эту книгу. Он был глубоко интеллигентным и мудрым человеком, и я счастлив, что имел возможность следовать по его стопам. Наконец, я искренне признателен сообществу OpenCV за благосклонное отношение к первому изданию этой книги и за проявленное терпение, когда выход этого издания задерживался из-за многочисленных волнительных, но несколько отвлекающих начинаний.

На подготовку этого издания книги ушло много времени. За прошедшие годы мне довелось поработать с десятками компаний в качества консультанта, помогающего выстроить нужную им технологию. Выступая в роли члена правления с решающим или совещательным голосом, технического сотрудника, консультанта или основателя компании, я имел возможность наблюдать и восторгаться всеми аспектами процесса разработки новой технологии. Много лет я провел в компании Applied Minds, Inc., где занимался сначала созданием, а потом управлением отделом робототехники, а также в компании Applied Invention Corporation, отпочковавшейся от Applied Minds, в качестве коллегиального члена. Мне всегда доставляло удовольствие видеть, что OpenCV занимает центральное место в выдающихся проектах в самых разных областях: от здравоохранения и сельского хозяйства до авиации, обороны и национальной безопасности. Не меньшее удовольствие я испытывал, видя первое издание этой книги на столах сотрудников практически в каждой организации, где я работал. Технология, которую мы с Гэри использовали для создания Stanley, с тех пор стала неотъемлемой частью бесчисленных проектов – и не в последнюю очередь многих разрабатываемых сейчас проектов автомобилей без водителя. Любой из них, а быть может, и все сразу потенциально способны изменить и улучшить повседневную жизнь огромного числа людей. Какая радость – быть частью всего этого! Сколько выдающихся умов я повстречал за эти годы – и они говорили мне, какую роль сыграло первое издание в учебе, в преподавании, в выстраивании карьеры, в достигнутых успехах, – и каждый раз их слова приводили меня в состояние счастья и изумления. Я надеюсь, что новое издание продолжит служить всем вам и станет источником вдохновения для нового поколения ученых, инженеров и изобретателей.

Заканчивая последнюю главу этой книги, мы начинаем новые главы своей жизни работой в области робототехники, искусственного интеллекта, компьютерного зрения и т. д. Лично я глубоко благодарен всем тем, кто своей работой дал мне возможность сделать следующий шаг в жизни: учителям, наставникам и авторам книг. Я надеюсь, что новое издание нашей книги позволит и другим людям сделать очередной важный шаг в жизни. Хочется надеяться, что мы встретимся на этом пути!

Дополнения от Гэри...

Я начал работать над библиотекой OpenCV в 1999 году, поставив целью ускорить развитие компьютерного зрения и искусственного интеллекта и дать в руки каждому желающему инфраструктуру, которую в то время можно было встретить только в ведущих лабораториях. Мало что в жизни происходит по намеченному плану, и я благодарен судьбе, что эта мечта стала явью спустя долгих 17 (!) лет. Это случилось благодаря помощи со стороны многочисленных друзей и коллег – слишком многочисленных, чтобы их можно было перечислить здесь[1]. Но я все равно хочу выделить группу из России, с которой я начал работать в Intel. Они создали успешную компанию в области компьютерного зрения (Itseez.com), которая в конечном итоге была выкуплена Intel. Мы начинали как коллеги, но в итоге стали добрыми друзьями.

Когда в доме три подростка, жена, Соня Бродски, делает для выхода книги больше, чем я сам. Я очень люблю ее и бесконечно благодарен. Детей я тоже люблю, но не могу сказать, что они ускорили выход книги. :)

К работе над этим вариантом книги я приступил, когда трудился в стартапе Industrial Perception Inc., который помогал создавать и который был продан Google в 2013 году. С тех пор работа велась урывками, иногда по выходным, иногда поздними вечерами. И надо же, на дворе уже 2016 год – как летит время, когда работы невпроворот! Некоторые размышления в конце главы 23 навеяны природой интеллекта роботов, с которым я столкнулся, работая над PR2, двуруким роботом, сконструированным компанией Willow Garage, а также над проектом Stanley в Стэнфорде – роботом, выигравшим приз в 2 миллиона долларов на соревнованиях автомобилей-роботов DARPA Grand Challenge.

Заканчивая эту книгу, мы надеемся встретить вас в стартапах, исследовательских лабораториях, в академических кругах, на конференциях и семинарах, в ректоратах университетов и среди участников интересных коммерческих проектов. Не сочтите за труд черкнуть пару строк и рассказать, над какой крутой штукой работаете. Я создал OpenCV, чтобы поддержать и ускорить развитие компьютерного зрения и искусственного интеллекта для общего блага; все остальное делать вам. Мы живем в творческом мире, где один придумывает горшок, другой превращает его в барабан и т. д. Творите! Используйте OpenCV, чтобы создать что-то необыкновенно хорошее для всех нас!

---

[1] Сейчас у нас множество соавторов, вы можете убедиться в этом, пролистав список обновлений в журналах по адресу https://github.com/opencv/opencv/wiki/ChangeLog. Новых алгоритмов и приложений стало так много, что мы размещаем лучшие в самоподдерживаемых автономных модулях в каталоге *opencv_contrib*.

Глава 1

Общие сведения

Что такое OpenCV?

OpenCV [OpenCV] – это библиотека компьютерного зрения с открытым исходным кодом (см. http://opensource.org), доступная на сайте http://opencv.org. В 1999 году Гэри Брэдски [Bradski], работавший в корпорации Intel, начал проект OpenCV в надежде ускорить развитие компьютерного зрения и искусственного интеллекта, предоставив основательную инфраструктуру всем работающим в этой области. Библиотека написана на C и C++ и работает на платформах Linux, Windows и Mac OS X. Активно ведется работа по созданию интерфейсов к Python, Java, MATLAB и другим языкам, а также по переносу библиотеки на платформы Android и iOS для мобильных приложений. На протяжении многих лет разработку OpenCV поддерживали корпорации Intel и Google, а особенно компания Itseez [Itseez] (недавно приобретенная Intel), которая проделала львиную долю работы на ранних этапах развития. Позже к сопровождению всегда открытого и свободного проекта OpenCV.org [OpenCV] присоединилась компания Arraiy [Arraiy].

При проектировании OpenCV закладывалась максимальная вычислительная эффективность с упором на приложения реального времени. Она написана на оптимизированном C++ и может пользоваться преимуществами многоядерных процессоров. Если вы хотите добиться еще большего быстродействия на платформах с архитектурой Intel, то можете купить разработанные Intel библиотеки *Integrated Performance Primitives (IPP)* [IPP], включающие оптимизированные низкоуровневые процедуры для многих алгоритмов. OpenCV автоматически использует подходящую библиотеку IPP, если она установлена. Начиная с версии OpenCV 3.0, Intel передала команде и сообществу OpenCV бесплатное подмножество IPP (получившее название IPPICV), которое по умолчанию встроено в OpenCV и обеспечивает ускорение.

Одна из целей OpenCV – предоставить простую для использования инфраструктуру компьютерного зрения, которая позволила бы быстро создавать относительно сложные приложения. Библиотека OpenCV насчитывает свыше 500 функций, охватывающих многие области компьютерного зрения, в т. ч. контроль качества продукции, медицинские изображения, безопасность, пользовательские интерфейсы, калибровку камеры, стереозрение и робототехнику. Поскольку компьютерное зрение и машинное обучение часто идут рука об руку, в OpenCV включена также универсальная библиотека машинного обучения (модуль ML). Акцент в этой подбиблиотеке сделан на статистическом распознавании образов и кластеризации. Модуль ML исключительно полезен для основной миссии OpenCV, но вместе с тем обладает достаточной общностью для решения любых задач машинного обучения.

Кто использует OpenCV?

Многие специалисты в области теоретической информатики и программисты-практики знакомы с той или иной гранью компьютерного зрения, но немногие знают обо всех способах его применения. Так, большинство людей слышало об использовании компьютерного зрения в охранных системах и многие в курсе того, что оно все чаще применяется для обработки изображений и видео в вебе. Кто-то видел, как компьютерное зрение интегрируется в игровые интерфейсы. Еще меньше народу понимают, что в большинстве изображений, полученных с помощью аэрофотосъемки, а также в панорамах (как в «Просмотре улиц» от Google) широко используется калибровка камеры и методы сшивки изображений. Кое-кто знает о нишевых применениях в области мониторинга безопасности, беспилотных летательных аппаратах и анализе биомедицинских данных. Но лишь немногие осознают, насколько глубоко компьютерное зрение проникло в производство: практически все товары массового производства на каком-то этапе проходят автоматический контроль с помощью системы компьютерного зрения.

Лицензия, по которой распространяется OpenCV, допускает использование OpenCV полностью или частично в коммерческих продуктах. Вы не обязаны раскрывать исходный код своего продукта или делать внесенные вами усовершенствования общедоступными, хотя мы на это надеемся. Отчасти из-за таких либеральных условий вокруг OpenCV сложилось обширное сообщество, включающее и представителей крупных компаний (IBM, Microsoft, Intel, SONY, Siemens, Google – и это далеко не все) и исследовательских центров (Стэнфордский университет, Массачусетский технологический институт, университет Карнеги-Меллон, Кембридж и государственный институт исследований в информатике и автоматике – INRIA). В группах Yahoo существует форум (https://groups.yahoo.com/neo/groups/OpenCV/info), насчитывающий свыше 50 000 участников. OpenCV популярна во всем мире, но особенно крупные сообщества сложились в Китае, Японии, России, Европе и Израиле.

С момента выхода альфа-версии в январе 1999 года OpenCV нашла применение во многих приложениях, продуктах и научных исследованиях, например: сшивка изображений для получения спутниковых и веб-карт, совмещение результатов сканирования изображений, подавление шумов на медицинских изображениях, анализ объектов, системы наблюдения и обнаружения вторжений, автоматический мониторинг в системах технической безопасности, системы контроля производственных процессов, калибровка камер, военные приложения, а также беспилотные воздушные, наземные и подводные транспортные средства. Методы компьютерного зрения применяются даже к анализу спектрограмм для распознавания звука и музыки. OpenCV лежала в основе системы зрения робота Stanley, созданного в Стэнфордском университете и завоевавшего учрежденный агентством DARPA приз в 2 миллиона долларов на соревнованиях автомобилей-роботов в пустыне [Thrun06].

Что такое компьютерное зрение?

Компьютерным зрением[1] называется преобразование данных, полученных от фото- или видеокамеры, результатом которого является некоторое решение или новое

[1] Компьютерное зрение – обширная дисциплина. В этой книге излагаются лишь основы, но мы рекомендуем также другие учебники, в т. ч. [Trucco98] – простое введение, [Forsyth03] – исчерпывающий справочник и [Hartley06] и [Faugeras93] – обсуждение принципов трехмерного зрения.

представление. Преобразование производится ради достижения определенной цели. Исходные данные могут включать контекстную информацию, например «камера установлена в автомобиле» или «лазерный дальномер показывает, что объект находится на расстоянии 1 метр». Решение может иметь вид «на этой сцене присутствует человек» или «на этом слайде видно 14 опухолевых клеток». Под новым представлением может пониматься преобразование цветного изображения в полутоновое или исключение собственного движения камеры из последовательности изображений.

Поскольку человек воспринимает мир в том числе глазами, легко впасть в ошибку, думая, что компьютерное зрение – это просто. Ну, в самом деле, трудно ли увидеть машину на изображении? Первое интуитивное ощущение может оказаться совершенно ложным. Человеческий мозг разделяет зрительный сигнал на много каналов, передающих различную информацию. В мозгу имеется система внимания, которая выделяет важные части изображения (независимо от задачи), подавляя изучение остальных. В визуальном потоке существует сильная обратная связь, которая до сих пор плохо изучена. Существуют распространенные ассоциативные входные сигналы от мышц и всех остальных органов чувств, которые позволяют мозгу выводить перекрестные ассоциации, основанные на накопленном за всю жизнь опыте. Такие контуры обратной связи в мозгу имеются на всех этапах обработки, включая и сами «аппаратные датчики» (глаза), которые механически управляют освещенностью с помощью зрачка и регулируют восприятие на поверхности сетчатки.

Но в системе машинного зрения компьютер получает набор чисел от камеры или с диска – и только. В большинстве случаев нет ни встроенного механизма распознавания образов, ни автоматической системы фокусировки и настройки диафрагмы, ни перекрестных ассоциаций с прожитым опытом. Системы зрения, как правило, наивны. На рис. 1.1 показана фотография автомобиля. Компьютер «видит» только матрицу чисел. К любому числу примешан довольно сильный шум, поэтому само по себе оно дает мало информации, однако эта матрица – все, что «видит» компьютер. А наша задача – преобразовать эту зашумленную числовую матрицу в образ «бокового зеркала». Рисунок 1.2 позволяет лучше понять, почему компьютерное зрение – такая трудная задача.

На самом деле в той постановке, о которой мы говорили до сих пор, задача не просто трудная, а формально неразрешимая. Имея двумерное (2D) представление трехмерного (3D) мира, нельзя однозначно восстановить 3D-сигнал. Формально такие некорректные задачи не имеют единственного решения. Одно и то же 2D-изображение может представлять бесконечное число 3D-сцен, даже если данные идеально точны. Но, как мы уже отмечали, данные искажены помехами. Искажения обусловлены изменчивостью внешней среды (погода, освещение, отражения, перемещения), несовершенством линз и механических устройств, конечностью времени интеграции сенсором (размытость из-за движения), электрическими шумами в датчике или других электронных приборах, а также артефактами сжатия после захвата изображения. И как при таких пугающих условиях получить хоть какой-то результат?

При проектировании реальной системы часто можно воспользоваться знаниями о контексте для преодоления ограничения зрительных сенсоров. Рассмотрим в качестве примера подвижный робот, который должен находить и собирать канцелярские степлеры в здании. Робот может учитывать, что стол – это объект, находящийся внутри офиса, и что степлеры обычно лежат на столах. Это дает неявное представление о размере – степлер должен помещаться на столе. Кроме того, это позволяет

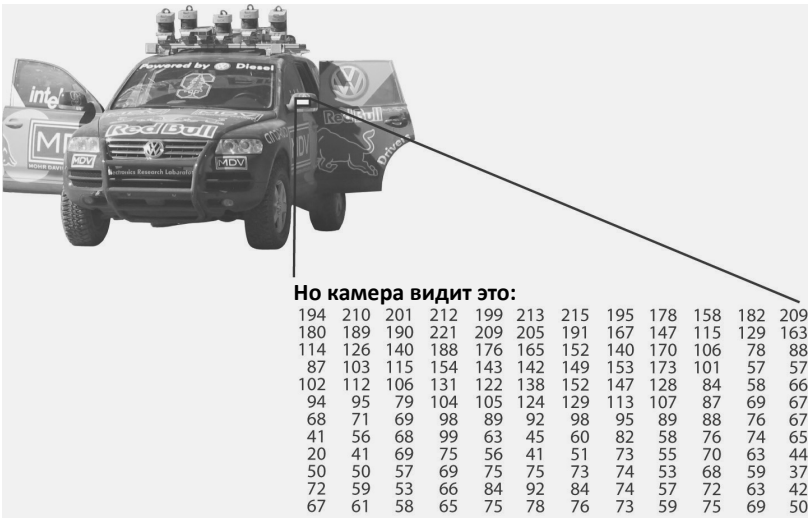

Рис. 1.1 ❖ Для компьютера боковое зеркало автомобиля – просто матрица чисел

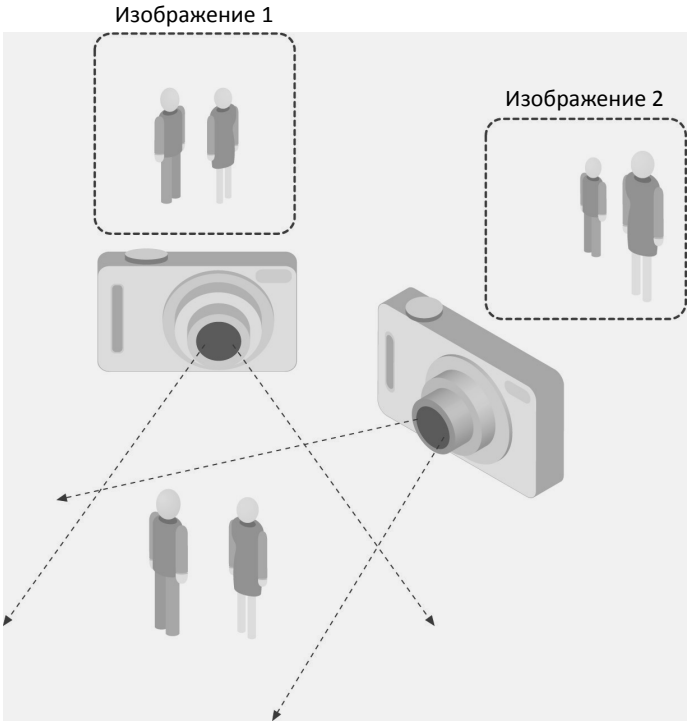

Рис. 1.2 ❖ Математически некорректная природа зрения: двумерное представление объектов может кардинально измениться в зависимости от точки съемки

исключить ложное «распознавание» степлеров в невозможных местах (например, на потолке или в окне). Робот может спокойно игнорировать 200-футовый рекламный аэростат в форме степлера, поскольку тому недостает фона с текстурой древесины, характерного для стола. С другой стороны, в задачах поиска изображений все изображения степлеров в базе данных – это фотографии реальных степлеров, поэтому большие размеры и прочие необычные конфигурации можно заранее исключить, исходя из предположения, что фотограф снимал настоящие степлеры нормального размера. Кроме того, фотограф обычно стремится поместить объект в центр изображения и в характерной для него ориентации. Поэтому зачастую в фотографиях, сделанных людьми, присутствует много неявной информации, включенной непреднамеренно.

Контекстную информацию можно моделировать и явно с помощью методов машинного обучения. Такие скрытые переменные, как размер, положение в пространстве и т. п., можно затем скоррелировать со значениями в размеченном обучающем наборе. Или можно попытаться измерить скрытые переменные с помощью дополнительных сенсоров. Используя лазерный дальномер для измерения глубины, мы сможем точно узнать размер объекта.

Следующая проблема – шумы, или помехи. Обычно для устранения шумов применяются статистические методы. Например, не всегда возможно распознать границу в изображении, просто сравнив точку с ее ближайшими соседями. Но если принять во внимание статистическое распределение в локальной области, то распознавание границ значительно упрощается. Истинная граница должна выглядеть как последовательность точек, для которых ориентация локальных областей согласована с соседями. Компенсировать помехи можно также, вычислив статистику по времени. Существуют и методы подавления помех, основанные на построении явных моделей, обучаемых по имеющимся данным. Например, поскольку природа искажения линзой хорошо изучена, нужно лишь обучить параметры простой полиномиальной модели, чтобы описать такие искажения, а значит, и почти полностью устранить их.

Действия или решения, которые система компьютерного зрения пытается принять на основе данных от камеры, зависят от конкретной задачи или цели. Возможно, мы хотим удалить шум или устранить повреждения изображения, чтобы охранная система поднимала тревогу, когда кто-то пробует перелезть через забор. Или нам нужно, чтобы система слежения подсчитывала, сколько людей пересекает границы определенной области в парке развлечений. Стратегии, применяемые программами компьютерного зрения робота, обходящего офисное здание, и стационарной камеры в охранной системе, различаются, потому что эти системы работают в разных контекстах и имеют разные цели. Вообще говоря, чем более ограничен контекст компьютерного зрения, тем вероятнее, что эти ограничения упрощают задачу, и тем надежнее будет конечное решение.

Цель OpenCV – предоставить базовые средства для решения задач компьютерного зрения. В некоторых ситуациях хватает высокоуровневой функциональности. Но даже если это не так, отдельные компоненты библиотеки достаточно полны, чтобы самостоятельно собрать из них решение почти любой задачи. И в этом случае существует несколько проверенных практикой способов применения библиотеки; все они начинаются с попытки задействовать в решении максимально большое число компонентов. Как правило, после создания первого приближения мы понимаем, в чем его слабые места, а затем исправляем их с помощью собственного кода и изобретательности (эту тактику характеризуют фразой «решай ту задачу, которая есть, а не ту, ко-

торую ты себе вообразил»). Первое черновое решение можно впоследствии использовать как эталон для оценки улучшений. Оставшиеся недостатки можно устранить, приняв в расчет контекст объемлющей системы, частью которой является созданное решение.

Истоки OpenCV

OpenCV появилась как плод исследований Intel, предпринятых для продвижения приложений, нуждающихся в мощном процессоре. С этой целью Intel начала много проектов, включая трассировку лучей в режиме реального времени и трехмерные видеостены. Один из авторов, Гэри Брэдски [*Bradski*], в то время работавший в Intel, был частым гостем в университетах и обратил внимание, что в некоторых ведущих учебных заведениях сформировались группы, например MIT Media Lab, с развитой и открытой для внутреннего пользования инфраструктурой компьютерного зрения – код передавался от одного студента другому, и таким образом каждый новый студент получал отправную точку для разработки собственного приложения. Вместо того чтобы заново изобретать базовые функции, студенты развивали то, что было создано до них.

Таким образом, OpenCV задумывалась как способ предоставить инфраструктуру компьютерного зрения в общее пользование. При поддержке группы разработки библиотеки повышения производительности (Performance Library Team) из Intel[1] была заложена основа OpenCV в виде уже написанного кода и спецификаций алгоритмов. Все это было отправлено членам российской группы разработчиков в составе Intel. Здесь и находятся истоки OpenCV: она была рождена в исследовательской лаборатории Intel в сотрудничестве с группой разработки библиотек повышения производительности, а реализацией и оптимизацией обязана опыту программистов из России.

Главным среди россиян был Вадим Писаревский, который руководил разработкой, кодировал сам и оптимизировал большую часть библиотеки, он и по сей день находится в центре большинства работ, связанных с OpenCV. В разработке ранних вариантов инфраструктуры ему помогал Виктор Ерухимов, а Валерий Курякин руководил российской лабораторией и всемерно поддерживал усилия коллег. С самого начала перед OpenCV было поставлено несколько целей.

- Способствовать исследованиям в области компьютерного зрения, предоставив не просто открытый, но и оптимизированный код базовой инфраструктуры. Хватит изобретать велосипед.
- Распространять знания о компьютерном зрении, предоставив единую инфраструктуру, которую можно надстраивать. Таким образом, код станет более понятным и переносимым.
- Продвигать коммерческие приложения на основе компьютерного зрения, бесплатно предоставив переносимый оптимизированный код – по лицензии, не требующей раскрытия или бесплатности самих коммерческих приложений.

Эти цели составляют ответ на вопрос «зачем». Наличие приложений компьютерного зрения вызывает потребность в более быстрых процессорах. Стимулирование перехода на более быстрые процессоры финансово более выгодно для Intel, чем продажа дополнительного программного обеспечения. Быть может, именно поэтому открытый и свободный код был написан в недрах компании по производству оборудо-

[1] И, прежде всего Шинна Ли.

вания, а не ПО. Иногда у производителя оборудования оказывается больше причин для инноваций в области софта.

В любом проекте ПО с открытым исходным кодом важно достичь критической массы, после чего проект поддерживает себя сам. На сегодняшний день OpenCV была скачана примерно 11 миллионов раз, и это число ежемесячно увеличивается в среднем на 160 000. В разработку OpenCV вносят вклад многие пользователи, а центр управления разработкой вышел за пределы Intel[1]. На рис. 1.3 показана история выпуска версий OpenCV. На OpenCV также оказали влияние бум «доткомов» и многочисленные смены руководства и направлений развития. Бывали периоды, когда над OpenCV не работал вообще никто из Intel. Но с появлением многоядерных процессоров и ряда новых приложений компьютерного зрения значение OpenCV вновь стало расти. Толчок дальнейшему развитию дал также быстрый прогресс в области робототехники. Превратившись в библиотеку с открытым исходным кодом, OpenCV несколько лет разрабатывалась при активном содействии Willow Garage, а теперь поддерживается фондом OpenCV (http://opencv.org/). В наши дни разработка ведется силами как фонда, так и нескольких публичных и частных организаций. Дополнительные сведения о будущем OpenCV см. в главе 23.

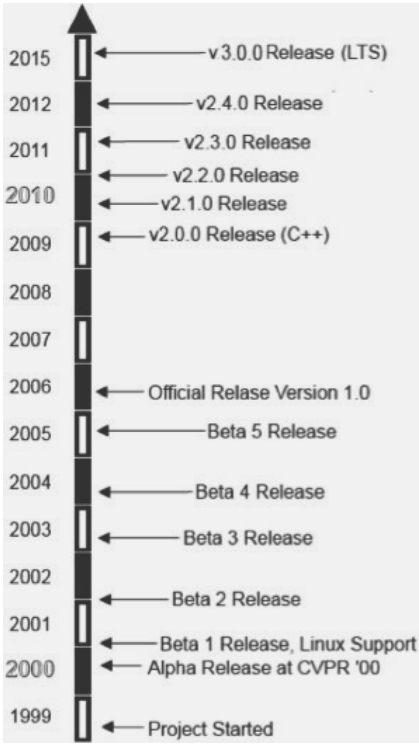

Рис. 1.3 ❖ История выпуска версий OpenCV

[1] На момент написания книги компания Willow Garage (http://www.willowgarage.com/) – исследовательский институт робототехники и инкубатор проектов, активно поддерживает общее сопровождение OpenCV и новые разработки в области робототехники.

Архитектура OpenCV

OpenCV состоит из нескольких уровней. На самом верху находится операционная система, в которой работает библиотека. Ниже расположены языковые привязки и примеры приложений. Еще ниже – предоставленный код из репозитория *opencv_contrib*, который содержит большую часть высокоуровневой функциональности. Далее – ядро OpenCV, а на самом нижнем уровне – различные аппаратные оптимизации, составляющие *уровень аппаратного ускорения* (hardware acceleration layer – HAL). Эта организация показана на рис. 1.4.

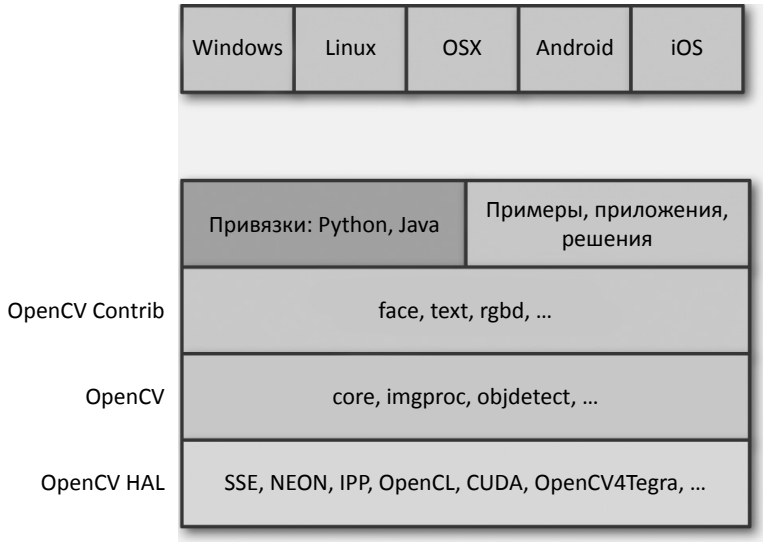

Рис. 1.4 ❖ Архитектурная диаграмма OpenCV и поддерживаемых операционных систем

Ускорение OpenCV с помощью IPP

При работе на процессорах Intel OpenCV пользуется не требующим лицензионных отчислений подмножеством библиотеки *Integrated Performance Primitives* (IPP) – IPP 8.x (IPPICV). IPPICV можно скомпоновать с OpenCV на этапе компиляции, и в таком случае она заменяет соответствующий низкоуровневый оптимизированный код, написанный на C (в cmake флаг `WITH_IPP=ON/OFF` по умолчанию равен `ON`). Использование IPP может дать заметное ускорение. На рис. 1.5 показан график относительного ускорения, обеспечиваемого IPP.

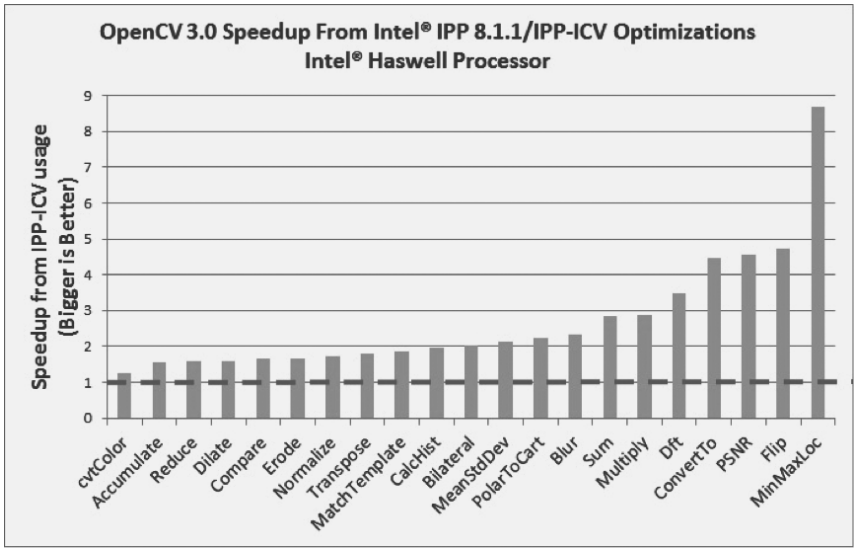

Рис. 1.5 ❖ Относительное ускорение при использовании OpenCV совместно с IPPICV на процессоре Intel Haswell

Кто является владельцем OpenCV?

Хотя инициатива создания OpenCV принадлежит Гэри Брэдски из Intel, целью библиотеки всегда было и остается способствование исследованиям и коммерческим применениям, в этом заключается ее миссия. Поэтому библиотека открыта и свободна, а ее код может включаться (полностью или частично) в другие приложения, коммерческие или исследовательские. Не требуется, чтобы само приложение было открытым или свободным. Мы не настаиваем, чтобы произведенные вами усовершенствования становились частью библиотеки, хотя и надеемся на это.

Скачивание и установка OpenCV

С главного сайта OpenCV (http://opencv.org/) можно скачать полный исходный код последней версии, а также многих предыдущих. Ссылки для скачивания находятся на странице http://opencv.org/downloads.html. Самую свежую версию всегда можно найти также на GitHub (https://github.com/opencv/opencv), где находится активно разрабатываемая ветка. Чтобы получить свежую функциональность верхнего уровня, скачайте и соберите приложения из каталога *opencv_contrib* [opencv_contrib] (https://github.com/opencv/opencv_contrib).

Установка

В настоящее время для управления версиями OpenCV используется Git, а для сборки – CMake[1]. Зачастую думать о сборке вам вообще не придется, потому что для многих платформ уже имеются готовые библиотеки. Но по мере обретения опыта вы

[1] Раньше разработчики OpenCV использовали для управления версиями Subversion, а для сборки – automake. Но эти времена давно прошли.

обязательно захотите перекомпилировать библиотеки с параметрами, отвечающими вашему приложению.

Windows

На странице http://opencv.org/downloads.html имеется ссылка для скачивания последней версии OpenCV для Windows. По ней вы получите исполняемый файл, представляющий собой самораспаковывающийся архив с готовым двоичным кодом OpenCV для разных версий Visual Studio. После этого почти все готово к использованию OpenCV[1].

Еще одна деталь – нужно будет добавить переменную среды `OPENCV_DIR`, чтобы было проще сообщить компилятору, где искать двоичные файлы OpenCV. Для этого нужно открыть окно команд и ввести[2]:

```
setx -m OPENCV_DIR D:\OpenCV\Build\x64\vc10
```

Для статической компоновки OpenCV больше ничего делать не надо. Если же вы собираетесь использовать *динамически загружаемые библиотеки* OpenCV (DLL), то надо еще сказать системе, где их искать. Для этого добавьте строку `%OPENCV_DIR%\bin` в состав путей к библиотекам. (В Windows 10 нужно щелкнуть правой кнопкой мыши по значку **Компьютер**, выбрать пункт **Свойства**, а затем нажать кнопку **Дополнительные параметры системы**. Наконец, выберите **Переменные среды** и добавьте путь к двоичным файлам OpenCV в переменную Path.)

В поставляемую версию OpenCV 3 уже включена библиотека IPP, поэтому вы получите ускорение на всех более-менее современных процессорах x86 и x64.

Можно также собрать OpenCV из исходного кода.

1. Запустите графический интерфейс CMake.
2. Укажите путь к дереву исходного кода OpenCV и каталогу сборки (они должны различаться!).
3. Два раза нажмите **Configure** (выберите подходящий генератор для Visual Studio или make-файлов MinGW, если пользуетесь компилятором MinGW), затем нажмите **Generate**.
4. Откройте сгенерированное решение в Visual Studio и соберите его. В случае MinGW следуйте инструкциям для Linux, приведенным ниже.

Linux

В состав комплекта поставки OpenCV для Linux не входят скомпилированные библиотеки из-за широкого разнообразия версий GCC и GLIBC в различных дистрибутивах (SuSE, Debian, Ubuntu и т. д.). Но часто OpenCV входит в сам дистрибутив. Если же это не так, то придется собрать библиотеки из исходного кода. Как и в случае Windows, начать следует со страницы http://opencv.org/downloads.html, но теперь ссылка ведет на сайт SourceForge, где находится архив с актуальным исходным кодом.

Для сборки библиотек и демонстрационных программ понадобится библиотека GTK+ версии 2.x или выше вместе с заголовками. Кроме того, необходим компилятор

---

[1] Важно знать, что в дистрибутив для Windows входят только выпускные, но не отладочные версии библиотек. Поэтому очень может статься, что до начала разработки вы захотите открыть файл решения и собрать библиотеки самостоятельно.

[2] Разумеется, точный путь зависит от места установки; например, если вы установили библиотеку на 32-разрядную машину, то путь будет содержать компонент `x64`, а не `x32`.

gcc со всеми пакетами для разработки, *cmake* и *libtbb* (библиотека Intel Thread Building Blocks) и факультативно библиотеки *zlib*, *libpng*, *libjpeg*, *libtiff* и *libjasper* вместе с файлами для разработки (т. е. версии пакетов с названиями, оканчивающимися на *-dev*).

Для работы привязок к Python понадобится версия Python 2.6 или более поздняя с установленными заголовками (пакет для разработки), а также библиотека NumPy. Кроме того, нужны библиотека *libavcodec* и другие библиотеки *libav** (с заголовками) из пакета *ffmpeg*.

Для этого установите пакеты *libav/ffmpeg*, включенные в дистрибутив, или скачайте *ffmpeg* с сайта http://www.ffmpeg.org. Библиотека *ffmpeg* распространяется по лицензии *Lesser General Public License* (LGPL), но для некоторых ее частей действует более ограничительная лицензия General Public License (GPL). Чтобы можно было использовать код в составе ПО, распространяемого не по лицензии GPL (в частности, OpenCV), соберите разделяемую библиотеку *ffmpg* и пользуйтесь ей:

```
$> ./configure --enable-shared
$> make
$> sudo make install
```

(При динамической компоновке LGPL-библиотеки распространять свой код по лицензии GPL не обязательно.) В результате будут созданы файлы */usr/local/lib/libavcodec.so.**, */usr/local/lib/libavformat.so.**, */usr/local/lib/libavutil.so.** и включаемые файлы в различных каталогах */usr/local/include/libav**.

Для сборки самой библиотеки нужно распаковать *.tar.gz*-файл, перейти в созданный каталог и выполнить следующие команды:

```
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install # не обязательно
```

Первая команда создает новый подкаталог, вторая делает его текущим. Третья команда говорит CMake, как сконфигурировать сборку. Показанные параметры годятся для начала, но есть и другие, которые включают различные возможности, определяют, какие примеры собирать, следует ли включать поддержку Python, поддержку CUDA GPU и т. д. По умолчанию конфигурационный скрипт *cmake* для сборки OpenCV пытается найти и использовать все сторонние библиотеки, какие сможет. Например, если он найдет CUDA SDK, то включит возможность ускорения с использованием графических процессоров. Последние два команды собирают библиотеку и устанавливают результаты в нужные места. Отметим, что для использования OpenCV в проектах, собираемых с помощью CMake, устанавливать ее не обязательно; достаточно указать путь к сгенерированному файлу *OpenCVConfig.cmake*. В примере выше этот файл окажется в каталоге *release*. Но если вы все-таки решите выполнить команду `sudo make install`, то файл *OpenCVConfig.cmake* будет помещен в каталог */usr/local/share/OpenCV*.

Как и в случае Windows, библиотека OpenCV, собранная в Linux, автоматически пользуется библиотекой IPP, если та установлена. Начиная с версии OpenCV 3.0, конфигурационный скрипт *cmake* автоматически скачивает и прикомпоновывает свободное подмножество IPP (IPPICV). Если вы хотите явно отключить IPP, то запускайте CMake с флагом `-D WITH_IPP=OFF`.

Mac OS X

Установка в системе OS X очень похожа на Linux, с тем отличием, что OS X поставляется с собственной средой разработки Xcode, которая включает почти все необходимое, кроме CMake; не понадобится GTK+, TBB, *libjpeg*, *ffmpeg* и т. д.:

- по умолчанию вместо GTK+ используется Cocoa;
- по умолчанию вместо *ffmpeg* используется QTKit;
- вместо TBB и OpenMP используется Grand Dispatch Central (GDC).

Шаги установки точно такие же. Если передать CMake флаг `-G Xcode`, то будет сгенерирован проект Xcode для сборки OpenCV (и ваших приложений). Это позволит удобно собирать и отлаживать код в среде Xcode.

Получение самой последней версии OpenCV из Git

OpenCV активно разрабатывается, ошибки исправляются быстро, если в отчет включены точное описание и код, позволяющий воспроизвести ошибку. Но официальные версии OpenCV выпускаются один или два раза в год. Если вы работаете над серьезным проектом, то, вероятно, хотите получать исправления ошибок и обновления сразу после их появления. Для этого нужен доступ к репозиторию OpenCV на GitHub.

Здесь не место для руководства по работе с Git. Если вам доводилось работать с другими проектами с открытым исходным кодом, то, наверное, вы уже знаете, что к чему. Если нет, почитайте книгу Jon Loeliger «Version Control with Git», вышедшую в издательстве O'Reilly (http://shop.oreilly.com/product/0636920022862.do). Командный клиент Git доступен для Linux, OS X и большинства производных от UNIX систем. Пользователям Windows мы рекомендуем программу TortoiseGit (https://tortoisegit.org/); пользователям OS X, возможно, понравится приложение SourceTree.

В Windows, чтобы получить последнюю версию OpenCV из репозитория Git, нужно зайти в каталог по адресу https://github.com/opencv/opencv.git.

В Linux достаточно такой команды:

```
git clone https://github.com/opencv/opencv.git
```

Дополнительная документация по OpenCV

Основная документация по OpenCV в формате HTML размещена на сайте http://opencv.org. Кроме того, ссылки на детальные пособия по различным темам опубликованы на странице http://docs.opencv.org/2.4.13/doc/tutorials/tutorials.html и в вики OpenCV (в настоящее время находится по адресу https://github.com/opencv/opencv/wiki).

Документация в комплекте поставки

В комплект поставки OpenCV 2.x входят полное справочное руководство и набор учебных пособий – все в формате PDF; загляните в каталог *opencv/doc*. Начиная с версии OpenCV 3.x документация в виде файлов не поставляется.

Онлайновая документация и вики

Как уже было сказано, на сайте http://opencv.org размещена подробная документация и вики. Документация организована в виде нескольких компонентов.

Справочное руководство
Сюда входит информация о функциях, их аргументах и немного о том, как ими пользоваться.

Учебные пособия
Пособий много; в них описано, как решать конкретные задачи. Есть пособия по базовым вещам, например как установить OpenCV или создать проект на разных платформах, а есть и по более сложным темам, например о вычитании фона в задаче обнаружения объектов.

Быстрый старт
Тщательно отобранное подмножество пособий, в которое включены только те, что помогут быстро начать работу на конкретных платформах.

Шпаргалка
Это единственный PDF-файл, который содержит великолепную сжатую справку почти по всей библиотеке. Спасибо Вадиму Писаревскому за эту отличную памятку, которую можно прикнопить у себя над столом.

Вики
В вики вы найдете все, что нужно, и многое сверх того. Здесь есть планы развития, новости, открытые вопросы, система отслеживания ошибок и бесчисленные обсуждения глубоких тем, например как стать соавтором OpenCV.

Вопросы и ответы
Обширный архив, содержащий тысячи заданных вопросов и ответов на них. Здесь вы можете задать свой вопрос сообществу или помочь другим, отвечая на вопросы.

Все это становится доступным при переходе по ссылке Documentation на домашней странице OpenCV.org. Из перечисленных замечательных ресурсов один заслуживает дополнительного обсуждения – справочное руководство. Оно разбито на несколько разделов, каждый из которых относится к одному библиотечному *модулю*. Список модулей со временем изменялся, но одно остается неизменным: модуль – основная организационная структура внутри библиотеки. Любая функция принадлежит одному модулю. Ниже перечислены модули, существующие на данный момент:

`core`
Это «ядро» библиотеки, сюда входят основные типы объектов и операции над ними.

`imgproc`
Модуль обработки изображений включает базовые преобразования, включая фильтры и подобные им сверточные операторы.

`highgui` (в OpenCV 3.0 разбит на `imgcodecs`, `videoio` и `highgui`)
Содержит функции пользовательского интерфейса, применяемые для вывода изображений или организации простого ввода. Можно считать, что это очень облегченный набор инструментов для создания оконного UI.

`video`
Содержит функции для чтения и записи видеопотоков.

`calib3d`
Содержит реализации алгоритмов калибровки одиночной камеры, а также многокамерных установок для стереосъемки.

`features2d`
Содержит алгоритмы обнаружения, описания и сопоставления особых точек.

`objdetect`
Содержит алгоритмы обнаружения конкретных объектов, например лиц или пешеходов. Можно обучить детектор обнаружению других объектов.

`ml`
Модуль машинного обучения представляет собой отдельную библиотеку и содержит разнообразные алгоритмы машинного обучения, реализованные так, чтобы было удобно работать с естественными структурами данных OpenCV.

`flann`
Акроним FLANN означает «Fast Library for Approximate Nearest Neighbors» (быстрая библиотека приближенной кластеризации по ближайшим соседям). В этой библиотеке собраны методы, которые вы вряд ли будете использовать напрямую, но которые используются функциями из других модулей для кластеризации больших наборов данных методом ближайших соседей.

`gpu` (в OpenCV 3.0 разбит на несколько модулей `cuda*`)
Содержит реализации большинства библиотечных функций из других модулей, оптимизированных для работы на графических процессорах, поддерживающих технологию CUDA. Существует также несколько функций, предназначенных только для работы на GPU. Некоторые из них дают отличные результаты, но требуют столько вычислительных ресурсов, что их реализация на обычном оборудовании не имеет смысла.

`photo`
Сравнительно новый модуль, содержащий средства, полезные для вычислительной фотографии.

`stitching`
Весь этот модуль посвящен реализации сложного конвейера для сшивки изображений. Эта функциональность включена в библиотеку недавно, но, как и модуль `photo`, представляет область, в которой ожидается быстрый прогресс.

`nonfree` (в OpenCV 3.0 перемещен в *opencv_contrib/xfeatures2d*)
OpenCV содержит несколько реализаций алгоритмов, которые запатентованы или обременены другими ограничениями (например, алгоритм SIFT). Они выделены в отдельный модуль, дабы подчеркнуть, что для использования их в коммерческом продукте необходимо проделать дополнительную работу.

`contrib` (в OpenCV 3.0 распределен по нескольким модулям *opencv_contrib*)
Содержит новую функциональность, которую еще только предстоит интегрировать в библиотеку.

`legacy` (в OpenCV 3.0 исключен)
Содержит старую функциональность, которая вообще должна быть исключена из библиотеки.

`ocl` (в OpenCV 3.0 исключен, заменен технологией T-API)
Это сравнительно новый модуль, который можно рассматривать как аналог модуля GPU, только он реализует открытый стандарт параллельного программирования Khronos OpenCL. В настоящее время он развит намного слабее, чем модуль GPU, но предполагается, что в будущем будет содержать реализации, способные работать на любом графическом процессоре или ином параллельном устройстве, поддерживающем стандарт Khronos (в отличие от модуля `gpu`, который рассчитан только на комплект инструментов CUDA компании NVidia и потому может работать лишь на графических процессорах NVidia).

Несмотря на постоянно улучшающееся качество онлайновой документации, есть одна вещь, для которой она не предназначена: объяснение реализованных алгоритмов и точного смысла их параметров. Цель этой книги – восполнить этот пробел, а также более глубоко описать основные строительные блоки библиотеки.

Репозиторий предоставленного кода OpenCV

Раньше OpenCV была монолитной библиотекой, но в версии 3.0 она разбита на две части: зрелая opencv и передовая функциональность в репозитории *opencv_contrib* [opencv_contrib]. Первая сопровождается постоянной командой OpenCV и содержит стабильный (по большей части) код, тогда как вторая, не столь зрелая, разрабатывается и сопровождается в основном силами сообщества; она может содержать части, не подпадающие под действие лицензии OpenCV, а также запатентованные алгоритмы.

Ниже перечислено несколько модулей из репозитория *opencv_contrib* (полный список на момент написания книги приведен в приложении B).

`Dnn`
Глубокие нейронные сети.

`face`
Распознавание лиц.

`text`
Выделение и распознавание текста, факультативно может использовать библиотеку распознавания символов с открытым исходным кодом Tesseract.

`rgbd`
Совместная обработка информации о цветности и карте глубины, полученной от Kinect и других сенсоров глубины (или просто найденной с помощью алгоритмов сопоставления стереоизображений).

`bioinspired`
Биокомпьютерное зрение (biologically inspired vision).

`ximgproc`, `xphoto`
Передовые алгоритмы обработки изображений и вычислительной фотографии.

`tracking`
Современные алгоритмы сопровождения объектов.

Скачивание и сборка предоставленных модулей

В Linux и OS X для скачивания *opencv_contrib* достаточно следующей команды:

```
git clone https://github.com/opencv/opencv_contrib.git
```

В Windows укажите этот адрес в TortoiseGit или аналогичном клиенте. После этого нужно будет переконфигурировать OpenCV с помощью CMake:

```
cmake -D CMAKE_BUILD_TYPE=Release \
  -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules ..
```

и собрать как обычно. Собранные предоставленные модули помещаются в тот же каталог, что и стандартные двоичные файлы OpenCV; чтобы ими воспользоваться, никакие дополнительные шаги не требуются.

Переносимость

OpenCV проектировалась как переносимая библиотека. Она изначально писалась, так чтобы ее можно было скомпилировать любым совместимым со стандартами компилятором C++. Поэтому использовались только стандартные возможности C и C++, что облегчает кросс-платформенную поддержку. В табл. 1.1 показано, на каких платформах OpenCV заведомо работает. Поддержка 32- и 64-разрядных архитектур Intel и AMD (x86, x64) наиболее зрелая, а поддержка ARM быстро улучшается. Что касается операционных систем, то OpenCV полноценно поддерживается в Windows, Linux, OS X, Android и iOS.

Если некоторая архитектура или ОС отсутствует в табл. 1.1, это еще не значит, что OpenCV не перенесена на нее. OpenCV перенесена практически на все коммерческие системы – от Amazon Cloud и 40-ядерного Intel Xeon Phi до Raspberry Pi и собак-роботов.

Таблица 1.1. Справка по переносимости OpenCV для версии 1.0

	x86/x64	AMD	Прочие: MIPS, PPC
Windows	SIMD, IPP, Parallel, I/O	SIMD, Parallel (3.0), I/O	–
Linux	SIMD, IPP, Parallel[a], I/O	SIMD, Parallel[a], I/O	Parallel[a], I/O*
Android	SIMD, IPP (3.0), Parallel[b], I/O	SIMD, Parallel[b], I/O	MIPS – базовая поддержка
OS X/iOS	SIMD, IPP (3.0), Parallel, I/O	SIMD, Parallel, I/O	–
Прочие: BSD, QNX, ...	SIMD	SIMD	

[a] Распараллеливание в Linux обеспечивается сторонней библиотекой или включением OpenMP.
[b] Распараллеливание в Android обеспечивается с помощью Intel TBB.

Ниже приведена расшифровка акронимов в табл. 1.1.

SIMD
Для повышения быстродействия используются векторные команды: SSE для x86/x64, NEON для ARM.

IPP
Доступна библиотека Intel IPP. Начиная с версии 3.0 имеется свободное и бесплатное специализированное подмножество IPP (IPPICV).

Parallel
Для распределения обработки между несколькими процессорными ядрами используется некая стандартная или сторонняя библиотека многопоточности.

I/O
Для захвата и записи видео можно использовать некий стандартный или сторонний API.

Резюме

В этой главе мы рассмотрели историю OpenCV с момента ее основания Гэри Брэдски из Intel в 1999 году до современного состояния, когда она поддерживается компанией Arraiy. Мы обсудили мотивы создания OpenCV и немного поговорили о том, что в нее входит. Было отмечено, что базовая библиотека, OpenCV, отделена от новой функциональности, вынесенной в репозиторий *opencv_contrib* (см. приложение B). Мы также рассказали о многочисленных ссылках на относящиеся к OpenCV материалы в сети. Было также описано, как скачать и установить OpenCV, и приведены сведения о ее переносимости.

Упражнения

1. Скачайте и установите последнюю версию OpenCV. Соберите ее в отладочном и выпускном режиме.
2. Скачайте и соберите последнюю стволовую версию OpenCV из Git.
3. Опишите, по меньшей мере, три неоднозначности, возникающие при преобразовании трехмерной сцены в двумерное представление. Как бы вы стали преодолевать эти неоднозначности?

Глава 2

Введение в OpenCV

Включаемые файлы

После установки библиотеки OpenCV и настройки среды программирования наша следующая задача – написать какой-нибудь интересный код. Но сначала обсудим включаемые файлы-заголовки. По счастью, заголовки отражают новую модульную структуру OpenCV, описанную в главе 1. Главный заголовок `.../include/opencv2/opencv.hpp` просто подтягивает заголовки всех модулей OpenCV:

`#include "opencv2/core/core_c.h"`
Старые структуры данных на C и арифметические функции.

`#include "opencv2/core/core.hpp"`
Новые структуры данных на C++ и арифметические функции.

`#include "opencv2/flann/miniflann.hpp"`
Функции приближенной кластеризации методом ближайших соседей.

`#include "opencv2/imgproc/imgproc_c.h"`
Старые функции обработки изображений, написанные на C.

`#include "opencv2/imgproc/imgproc.hpp"`
Новые функции обработки изображений, написанные на C++.

`#include "opencv2/video/photo.hpp"`
Алгоритмы обработки и восстановления фотографий.

`#include "opencv2/video/video.hpp"`
Функции видеотрекинга и сегментации фона.

`#include "opencv2/features2d/features2d.hpp"`
Поддержка сопровождения по двумерным признакам.

`#include "opencv2/objdetect/objdetect.hpp"`
Каскадный детектор лиц, латентный SVM, HoG, детектор плоских участков.

`#include "opencv2/calib3d/calib3d.hpp"`
Калибровка и стереозрение.

`#include "opencv2/ml/ml.hpp"`
Машинное обучение: кластеризация, распознавание образов.

`#include "opencv2/highgui/highgui_c.h"`
Старые написанные на C функции вывода изображений, ползунков, взаимодействия с помощью мыши, ввода-вывода.

`#include "opencv2/highgui/highgui.hpp"`
Новые написанные на C++ функции вывода изображений, ползунков, взаимодействия с помощью мыши, ввода-вывода.

`#include "opencv2/contrib/contrib.hpp"`
Код, предоставленный пользователями: распознавание человеческой кожи, нечеткий алгоритм сопровождения объектов методом сдвига среднего, спин-изображения, самоподобные признаки.

Можете включать только файл *opencv.hpp*, но это замедлит компиляцию, поскольку включаются объявления всех вообще функций OpenCV. Если вам нужны, к примеру, лишь функции обработки изображений, то лучше включить только файл *opencv2/imgproc/imgproc.hpp*. Все файлы-заголовки находятся в каталоге *.../modules*. Так, полный путь к файлу *imgproc.hpp* имеет вид *.../modules/imgproc/include/opencv2/imgproc/imgproc.hpp*. Аналогично исходный код самих функций находится в соответствующем каталоге *src*. Например, функция `cv::Canny()` из модуля `imgproc` находится в файле *.../modules/improc/src/canny.cpp*.

Зная о включаемых файлах, мы можем написать на C++ свою первую программу с использованием OpenCV.

Можно использовать также устаревший код, например: обнаружение пятен, распознавание лиц методом скрытых марковских моделей (HMM), алгоритм Condensation и объекты Eigen. Для этого нужно включить заголовок *opencv2/legacy/legacy.hpp*, который находится в файле *.../modules/legacy/include/opencv2/legacy/legacy.hpp*.

Ресурсы

В Интернете есть несколько хороших вводных презентаций на PowerPoint с обзорами OpenCV:

- высокоуровневый обзор библиотеки по адресу http://is.gd/niZvJu;
- обсуждение методов ускорения по адресу http://is.gd/ShvMZE;
- описание модулей по адресу http://is.gd/izlOrM.

Первая программа – вывод изображения на экран

OpenCV предоставляет функции для чтения файлов в самых разных графических форматах, а также изображений, снятых фото- или видеокамерой. Все они входят в состав комплекта инструментов HighGUI, включенного в состав OpenCV. Мы воспользуемся некоторыми из них для написания простой программы, которая открывает файл изображения и выводит его на экран.

Пример 2.1 ❖ Простая программа OpenCV, которая загружает изображение с диска и выводит его на экран

```
#include <opencv2/opencv.hpp>  // заголовок, подтягивающий все функции OpenCV

int main( int argc, char** argv ) {
  cv::Mat img = cv::imread(argv[1],-1);
```

```
  if( img.empty() ) return -1;
  cv::namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
  cv::imshow( "Example1", img );
  cv::waitKey( 0 );
  cv::destroyWindow( "Example1" );
  return 0;
}
```

Отметим, что все функции OpenCV находятся в пространстве имен `cv`. Для вызова функции нужно явно сообщить компилятору имя пространства имен, добавив префикс `cv::` к имени функции. Чтобы избавиться от такого многословия, можно воспользоваться директивой `using namespace cv;`, как показано в примере 2.2[1]. Эта директива означает, что компилятор должен предполагать, что имя функции может находиться в указанном пространстве имен. Обратите также внимание на включаемые файлы в примерах 2.1 и 2.2; в первом случае включается общий файл *opencv.hpp*, а во втором – только необходимый файл; это уменьшает время компиляции.

Пример 2.2 ❖ То же, что пример 2.1, но с использованием директивы «using namespace»

```
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

int main( int argc, char** argv ) {

  Mat img = imread( argv[1], -1 );

  if( img.empty() ) return -1;

  namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
  imshow( "Example1", img );
  waitKey( 0 );

  destroyWindow( "Example1" );

}
```

Если откомпилировать[2] и запустить эту программу из командной строки с одним аргументом, то изображение будет загружено в память и выведено на экран. Затем программа подождет нажатия любой клавиши, после чего закроет окно и завершится. Проанализируем каждую строку этого кода, чтобы понять, что делает программа.

```
cv::Mat img = cv::imread( argv[1], -1 );
```

---

[1] Но, поступая так, вы рискуете нарваться на конфликт с именами из других пространств имен. Если некая функция, скажем `foo()`, существует в пространствах имен `cv` и `std`, то необходимо указать, о какой именно идет речь, с помощью нотации `cv::foo()`или `std::foo()`. Всюду в этой книге, кроме примера 2.2, мы будем указывать префикс `cv::` для объектов из пространства имен OpenCV, поскольку это считается более правильным стилем программирования.

[2] Конечно, команды сборки зависят от платформы. В этой книге мы, вообще говоря, не будем обращать внимания на платформенно-зависимые детали, но сейчас все-таки покажем, как может выглядеть команда сборки в UNIX: `gcc -v example2_2.cpp -I/usr/local/include/ -L/usr/lib/ -lstdc++ -L/usr/local/lib -lopencv_highgui -lopencv_core - -o example2_2`. Обратите внимание, что различные компоненты библиотеки обычно компонуются отдельно. В примере 2.3 ниже, где мы подключим видео, нужно будет добавить еще `-lopencv_imgcodecs -lopencv_imgproc -lopencv_videoio -lopencv_video -lopencv_videostab`.

В этой строке загружается изображение[1]. Высокоуровневая функция `cv::imread()` определяет формат файла с указанным именем и автоматически выделяет память для структуры данных изображения. Отметим, что `cv::imread()` умеет читать файлы в различных форматах, включая BMP, DIB, JPEG, JPE, PNG, PBM, PGM, PPM, SR, RAS и TIFF. Функция возвращает структуру типа `cv::Mat`. Это структура данных OpenCV, с которой мы чаще всего будем иметь дело. Она используется для изображений любого вида: одноканальных, многоканальных, кодированных целыми числами и числами с плавающей точкой и т. д. В следующей строке

```
if( img.empty() ) return -1;
```

проверяется, было ли изображение успешно прочитано. Еще одна высокоуровневая функция `cv::namedWindow()` открывает на экране окно, в которое можно вывести изображение.

```
cv::namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
```

Эта функция из библиотеки HighGUI заодно присваивает окну имя (в данном случае `"Example1"`). Последующие функции будут ссылаться на это окно по имени.

Второй аргумент `cv::namedWindow()` задает свойства окна. Он может быть равен либо `0` (значение по умолчанию), либо `cv::WINDOW_AUTOSIZE`. В первом случае размер окна не зависит от размера изображения, а изображение масштабируется под размер окна. Во втором случае окно автоматически уменьшается или увеличивается, подстраиваясь под истинный размер изображения, но впоследствии пользователь может изменить размер окна.

```
cv::imshow( "Example1", img );
```

Имея изображение в виде структуры `cv::Mat`, мы можем отобразить его в существующем окне, вызвав `cv::imshow()`. Эта функция создает окно, если оно еще не существует (не было создано функцией `cv::namedWindow()`). После вызова `cv::imshow()` окно перерисовывается вместе с изображением, а его размер подстраивается под размер изображения, если при создании был задан флаг `cv::WINDOW_AUTOSIZE`.

```
cv::waitKey( 0 );
```

Функция `cv::waitKey()` приостанавливает программу в ожидании нажатия на клавишу. Если задан положительный аргумент, то программа будет ждать такое количество миллисекунд, а затем продолжит выполнение, если клавиша не была нажата. Если же аргумент равен `0` или отрицателен, то программа будет ждать нажатия клавиши бесконечно.

Структура `cv::Mat` устроена так, что память, отведенная под изображение, автоматически освобождается, когда программа покидает область видимости переменной, – так же, как в контейнерных классах из стандартной библиотеки шаблонов (Standard Template Library – STL). Автоматическое освобождение управляется внутренним счетчиком ссылок. Следовательно, в большинстве случаев программисту не нужно

---

[1] Правильно написанная программа должна была бы проверить существование аргумента `argv[1]` и в случае его отсутствия выдать пользователю сообщение об ошибке с указанием, что делать. В этой книге мы будем опускать подобные вещи, предполагая, что читатель понимает важность обработки ошибок.

думать о выделении и освобождении памяти для изображений, как то было при работе со структурой `IplImage` в версии OpenCV 1.0.

```
cv::destroyWindow( "Example1" );
```

Наконец, можно уничтожить само окно. Функция `cv::destroyWindow()` закрывает окно и освобождает занятую им память. В коротких программах этот шаг можно опускать. Но в длинных и сложных программах необходимо уничтожать окна, прежде чем они выйдут из области видимости, иначе произойдет утечка памяти.

Далее мы напишем очень простую – почти такую же простую, как эта, – программу для чтения и отображения видеофайла. А затем начнем производить различные операции с изображениями.

Вторая программа – видео

OpenCV позволяет воспроизводить видео почти с такой же легкостью, как отдельное изображение. Единственное отличие – цикл чтения кадров; кроме того, нужен какой-то способ выйти из цикла, если фильм нам наскучит.

Пример 2.3 ❖ Простая программа OpenCV для воспроизведения видеофайла с диска

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

int main( int argc, char** argv ) {

  cv::namedWindow( "Example3", cv::WINDOW_AUTOSIZE );
  cv::VideoCapture cap;
  cap.open( string(argv[1]) );

  cv::Mat frame;
  for(;;) {
    cap >> frame;
    if( frame.empty() ) break;         // фильм кончился
    cv::imshow( "Example3", frame );
    if( cv::waitKey(33) >= 0 ) break;
  }

  return 0;
}
```

Здесь функция `main()` начинается с обычного создания именованного окна (в данном случае `"Example3"`). Затем создается объект захвата видео типа `cv::VideoCapture`. Этот объект умеет открывать и закрывать видеофайлы любых типов, поддерживаемых библиотекой *ffmpeg*.

```
cap.open(string(argv[1]));
cv::Mat frame;
```

Объекту захвата передается строка, содержащая полный путь к видеофайлу. После открытия файла объект будет содержать всю информацию о считываемом видео, включая и информацию о состоянии. При таком создании объект `cv::VideoCapture` инициализируется первым кадром видео. Затем программа создает объект `cv::Mat frame`, в котором будут храниться кадры видео.

```
cap >> frame;
if( frame.empty() ) break;
cv::imshow( "Example3", frame );
```

Внутри цикла for из видеофайла последовательно читаются кадры с помощью потокового объекта захвата. Программа проверяет, были ли прочитаны данные – if(frame.empty()) – и если нет, то завершается. Если очередной кадр успешно прочитан, то он отображается путем вызова cv::imshow().

```
if( cv::waitKey(33) >= 0 ) break;
```

После отображения кадра мы ждем 33 мс[1]. Если за это время пользователь нажмет какую-нибудь клавишу, то мы выходим из цикла чтения. В противном случае по прошествии 33 мс мы переходим к следующей итерации. После выхода из цикла вся выделенная память автоматически освобождается.

Перемотка вперед и назад

Теперь можно немного поэкспериментировать с нашими игрушечными программами и посмотреть, что еще есть в нашем распоряжении. Первое, чего не хватает проигрывателю из примера 2.3, – возможности быстро перемещаться по видео. Поэтому на следующем шаге мы добавим ползунок, который предоставит такую возможность. Заодно мы позволим пользователю переходить в режим покадрового просмотра нажатием клавиши **S** и возвращаться в обычный режим нажатием клавиши **R**. После перехода в новое место видео с помощью ползунка мы автоматически будем устанавливать покадровый режим.

В библиотеке HighGUI есть ряд средств для работы с изображениями и видео, помимо простых функций отображения, с которыми мы уже познакомились. Одно из них – ползунок – позволяет легко переходить из одного места видео в другое. Чтобы создать ползунок, нужно вызывать функцию cv::createTrackbar() и указать, в каком окне ползунок должен появиться. А для получения требуемой функциональности нам понадобится функция обратного вызова, которая и произведет перемотку.

Пример 2.4 ❖ Добавление в окно просмотра ползунка для перемещения по видеофайлу

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <fstream>

using namespace std;

int g_slider_position = 0;
int g_run = 1, g_dontset = 0; // начинаем в режиме покадрового просмотра
cv::VideoCapture g_cap;
```

[1] Можно ждать сколько угодно. В данном случае мы предполагаем, что было бы правильно воспроизводить видео с частотой 30 кадров в секунду и дать пользователю возможность прерывать воспроизведение между кадрами (поэтому мы делаем паузу на 33 мс после каждого кадра и ждем нажатия на клавишу). На практике лучше узнать из структуры cv::VideoCapture истинную частоту кадров (подробнее см. главу 8).

```
void onTrackbarSlide( int pos, void *) {

  g_cap.set( cv::CAP_PROP_POS_FRAMES, pos );
  if( !g_dontset )
    g_run = 1;
  g_dontset = 0;

}

int main( int argc, char** argv ) {

  cv::namedWindow( "Example2_4", cv::WINDOW_AUTOSIZE );
  g_cap.open( string(argv[1]) );
  int frames = (int) g_cap.get(cv::CAP_PROP_FRAME_COUNT);
  int tmpw = (int) g_cap.get(cv::CAP_PROP_FRAME_WIDTH);
  int tmph = (int) g_cap.get(cv::CAP_PROP_FRAME_HEIGHT);
  cout << "В виде " << frames << " кадров размером("
       << tmpw << ", " << tmph << ")." << endl;

  cv::createTrackbar("Position", "Example2_4", &g_slider_position, frames,
                onTrackbarSlide);

  cv::Mat frame;
  for(;;) {

    if( g_run != 0 ) {

      g_cap >> frame; if(frame.empty()) break;
      int current_pos = (int)g_cap.get(cv::CAP_PROP_POS_FRAMES);
      g_dontset = 1;

      cv::setTrackbarPos("Position", "Example2_4", current_pos);
      cv::imshow( "Example2_4", frame );

      g_run-=1;

    }

    char c = (char) cv::waitKey(10);
    if( c == 's' ) // покадровый режим
      {g_run = 1; cout << "Покадровый режим, run = " << g_run << endl;}
    if( c == 'r' ) // непрерывный режим
      {g_run = -1; cout << "Непрерывный режим, run = " << g_run <<endl;}
    if( c == 27 )
      break;

  }
  return(0);

}
```

По существу, мы добавили глобальную переменную, описывающую положение ползунка, и функцию обратного вызова, которая обновляет эту переменную и перемещает позицию чтения в видео. Одним вызовом мы и создаем ползунок, и присоединяем к нему функцию обратного вызова[1]. Теперь рассмотрим детали, начав с глобальных переменных.

---

[1] Отметим, что некоторые кодировки AVI и mpeg не допускают обратной перемотки видео.

```
int g_slider_position = 0;
int g_run             = 1;
int g_dontset         = 0;             // начинаем в режиме покадрового просмотра
VideoCapture g_cap;
```

Сначала определяется глобальная переменная `g_slider_position`, в которой запоминается положение ползунка. Функции обратного вызова необходим также доступ к объекту захвата `g_cap`, поэтому мы делаем его глобальным. Будучи ответственными разработчиками, мы хотим, чтобы наш код был понятным, поэтому принимаем соглашение – добавлять префикс `g_` к имени любой глобальной переменной. Попутно создается еще одна глобальная переменная, `g_run`, в которой запоминается количество новых кадров (если значение отлично от 0). Положительное число показывает, сколько кадров показано перед остановкой, отрицательное означает, что система работает в режиме непрерывного просмотра.

Во избежание недоразумений, когда пользователь переходит в новое место видео, щелкнув по ползунку, мы переходим в режим покадрового просмотра, присваивая `g_run = 1`. Но при этом возникает тонкая проблема: мы хотим, чтобы положение ползунка изменялось по мере просмотра видео. Для этого мы вызываем из `main` функцию `cv::setTrackbarPos`, которая обновляет положение ползунка после каждого кадра. Но вызов этой функции приводит к вызову функции обратного вызова ползунка, а нам не нужно, чтобы такие программные вызовы переключали просмотр в покадровый режим. Чтобы воспрепятствовать этому, мы завели еще одну глобальную переменную `g_dontset`, которая позволяет изменять положение ползунка без перехода в покадровый режим.

```
void onTrackbarSlide(int pos, void *) {

  g_cap.set(cv::CAP_PROP_POS_FRAMES, pos);

  if( !g_dontset )
    g_run = 1;
  g_dontset = 0;

}
```

Теперь разберемся с функцией обратного вызова, которая вызывается, когда пользователь двигает ползунок. Ей передается 32-разрядное целое `pos`, описывающее новое положение ползунка. Запрошенное положение мы передаем функции `g_cap.set()`, которая перематывает видео. В предложении `if` мы переводим программу в покадровый режим, но только в том случае, когда функция была вызвана в ответ на действие пользователя, а не в результате вызова `cv::setTrackbarPos()` из `main` (в последнем случае устанавливается флаг `g_dontset`).

Обращения к `g_cap.set()`, а равно и к парной функции `g_cap.get()` будут часто встречаться нам в дальнейшем. Эти функции устанавливают (или читают) различные свойства объекта `cv::VideoCapture`. В данном случае мы передаем аргумент `cv::CAP_PROP_POS_FRAMES`, который означает, что мы хотим установить позицию чтения, измеряемую в кадрах[1].

---

[1] Поскольку HighGUI – библиотека цивилизованная, при запросе новой позиции видео она автоматически разбирается со случаем, когда запрошенный кадр не является опорным; тогда она начнет с предыдущего опорного кадра и быстро перемотает видео к нужному кадру, не обременяя нас такими деталями.

```
int frames = (int) g_cap.get(cv::CAP_PROP_FRAME_COUNT);
int tmpw = (int) g_cap.get(cv::CAP_PROP_FRAME_WIDTH);
int tmph = (int) g_cap.get(cv::CAP_PROP_FRAME_HEIGHT);
cout << "В видео " << frames << " кадров размера ("
     << tmpw << ", " << tmph << ")." << endl;
```

В основе своей программа мало чем отличается от примера 2.3, поэтому сосредоточимся на том, что мы добавили. Прежде всего после открытия видеофайла мы обращаемся к функции `g_cap.get()`, чтобы узнать, сколько в нем кадров, а также ширину и высоту изображений, составляющих видео. Эти числа распечатываются. Число кадров нам нужно, чтобы откалибровать ползунок (на следующем шаге).

```
createTrackbar("Position", "Example2_4", &g_slider_position, frames, onTrackbarSlide);
```

Затем мы создаем сам ползунок. Функция `cv::createTrackbar()` позволяет сопоставить ползунку метку[1] (в данном случае `Position`) и указать, в какое окно поместить ползунок. Следующие аргументы – переменная, связываемая с ползунком, максимальное значение ползунка (количество кадров видео) и функция обратного вызова (или `NULL`, если она нам не нужна), вызываемая при изменении положения ползунка.

```
if( g_run != 0 ) {

    g_cap >> frame; if(!frame.data) break;
    int current_pos = (int)g_cap.get(cv::CAP_PROP_POS_FRAMES);
    g_dontset = 1;

    cv::setTrackbarPos("Position", "Example2_4", current_pos);
    cv::imshow( "Example2_4", frame );

    g_run-=1;

}
```

В цикле `for` мы не только читаем и отображаем кадр видео, но и получаем текущую позицию кадра, устанавливаем флаг `g_dontset`, чтобы следующий обратный вызов ползунка не перевел нас в покадровый режим, а затем вызываем функцию `cv::setTrackbarPos()`, которая изменяет видимое положение ползунка (и приводит к обратному вызову). Глобальная переменная `g_run` уменьшается на единицу, и в результате мы либо остаемся в покадровом режиме, либо продолжаем непрерывное воспроизведение – в зависимости от предыдущего состояния, которое пользователь задал нажатием клавиши.

```
    char c = (char) cv::waitKey(10);
    if( c == 's' )                    // покадровый режим
      {g_run = 1; cout << "Покадровый режим, run = " << g_run << endl;}
    if( c == 'r' )                    // непрерывный режим
      {g_run = -1; cout << "Непрерывный режим, run = " << g_run <<endl;}
    if( c == 27 )
      break;
```

---

[1] Поскольку библиотека HighGUI облегченная, `cv::createTrackbar()` не различает имя ползунка и метку, которая отображается рядом с ним на экране. Вы, вероятно, уже заметили, что и функция `cv::namedWindow()` показывает имя окна в качестве метки этого окна в пользовательском интерфейсе.

В конце цикла for мы смотрим, какую клавишу нажал пользователь. Если S, то переходим в покадровый режим (переменной g_run присваивается значение 1, которое позволяет читать по одному кадру). Если R, то мы переходим в режим непрерывного воспроизведения (g_run присваивается значение -1, так что при дальнейшем уменьшении она будет оставаться отрицательной для любого мыслимого размера видео). Наконец, если нажата клавиша **Esc**, то программа завершается. Снова отметим, что в коротких программах мы опускаем шаг освобождения памяти, занятой окном, – вызов cv::destroyWindow().

Простое преобразование

Отлично, с помощью OpenCV мы создали свой видеопроигрыватель, не так уж сильно отличающийся от бесчисленных уже существующих проигрывателей. Но нас-то интересует компьютерное зрение, поэтому надо бы сделать что-то в этом направлении. Во многих задачах, связанных со зрением, к видеопотоку применяются различные фильтры. Модифицируем программу, так чтобы она применяла к каждому кадру простую операцию.

К числу самых простых относится операция *сглаживания*, которая уменьшает количество информации в изображении, сворачивая его с гауссовым или еще каким-то ядром. В OpenCV такие преобразования свертки делаются совсем просто. Для начала создадим новое окно с именем "Example2_5-out", в котором будет показывать результаты обработки. Затем вызовем функцию cv::imshow(), чтобы показать очередной захваченный кадр в окне ввода, а потом вычислим сглаженное изображение и покажем его в окне вывода.

Пример 2.5 ❖ Загрузка и сглаживание изображения перед выводом на экран

```
#include <opencv2/opencv.hpp>

void example2_5( const cv::Mat & image ) {

  // Создаем несколько окон для показа входного
  // и выходного изображений.
  //
  cv::namedWindow( "Example2_5-in", cv::WINDOW_AUTOSIZE );
  cv::namedWindow( "Example2_5-out", cv::WINDOW_AUTOSIZE );

  // Показываем входное изображение
  cv::imshow( "Example2_5-in", image );

  // Создаем объект для хранения сглаженного изображения
  cv::Mat out;

  // Сглаживаем
  // ( Примечание: можно использовать  GaussianBlur(), blur(), medianBlur()
  // или bilateralFilter(). )
  //
  cv::GaussianBlur( image, out, cv::Size(5,5), 3, 3);
  cv::GaussianBlur( out, out, cv::Size(5,5), 3, 3);

  // Показываем сглаженное изображение в окне вывода
  cv::imshow( "Example2_5-out", out );

  // Ждем нажатия клавиши, окна уничтожатся автоматически
  cv::waitKey( 0 );
}
```

Первое обращение к `cv::imshow()` такое же, как в предыдущем примере. Далее мы выделяем память еще для одной структуры изображения. Объект `cv::Mat` упрощает нам жизнь; достаточно просто создать выходную матрицу `out`, а уж изменять свои размеры, выделять и освобождать память она будет автоматически по мере необходимости. Чтобы прояснить этот момент, мы далее используем даный объект в двух обращениях к функции `cv::GaussianBlur()`. Сначала исходное изображение размывается гауссовым сверточным фильтром размера 5×5, и результат записывается в `out`. Размеры гауссова ядра всегда должны быть нечетными, поскольку это ядро (его размер задан с помощью объекта `cv::Size(5,5)`) вычисляется относительно центрального пикселя в указанной области. В следующем вызове `cv::GaussianBlur()` переменная `out` используется и для входа, и для выхода, и в этом случае временная область памяти выделяется автоматически. Отображается последнее, дважды размытое изображение, после чего программа ждет нажатия клавиши, а затем завершается. Выделенная память освобождается, когда переменные выходят из области видимости.

Не столь простое преобразование

Все это прекрасно, мы научились делать интересные вещи. В примере 2.5 у гауссова размытия не было конкретной цели. А теперь воспользуемся функцией, которая применяет гауссово размытие для *уменьшения разрешения* изображения вдвое [Rosenfeld80]. Если проделать эту операцию несколько раз, то получится *масштабируемое пространство* (scale space), или *пирамида изображений*, которая часто используется в компьютерном зрении для изменения масштаба сцены или объекта.

Для читателей, знакомых с теорией обработки сигналов и теоремой отсчетов Найквиста–Шеннона[1] [Shannon49], отметим, что «уменьшение разрешения», или субдискретизация сигнала (в данном случае создание изображения путем отсчета каждого второго пикселя) эквивалентно свертке с последовательностью импульсных функций (можете считать их «пиками»). При таком отсчете в результирующий сигнал (изображение) добавляются высокие частоты. Чтобы избежать этого, мы хотим сначала наложить на сигнал высокочастотный фильтр, срезающий частоты, так чтобы все они оказались ниже частоты дискретизации. В OpenCV для гауссова размытия и субдискретизации существует функция `cv::pyrDown()`, демонстрируемая ниже.

Пример 2.6 ❖ Использование `cv::pyrDown()` для создания изображения, вдвое меньшего исходного

```
#include <opencv2/opencv.hpp>

int main( int argc, char** argv ) {

  cv::Mat img1,img2;

  cv::namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
  cv::namedWindow( "Example2", cv::WINDOW_AUTOSIZE );

  img = cv::imread( argv[1] );
  cv::imshow( "Example1", img1 );

  cv::pyrDown( img1, img2);
  cv::imshow( "Example2", img2 );
```

[1] В отечественной литературе ее называют теоремой Котельникова. – *Прим. перев.*

```
  cv::waitKey(0);

  return 0;

}
```

Теперь рассмотрим похожий, но чуть более сложный пример с использованием *детектора границ Кэнни* [Canny86] `cv::Canny()`. Детектор границ порождает изображение такого же размера, как исходное, но для записи результата ему нужно одноканальное изображение. Поэтому мы сначала преобразуем исходное изображение в полутоновое (одноканальное), воспользовавшись функцией `cv::cvtColor()` с флагом `cv::COLOR_BGR2GRAY`.

Пример 2.7 ❖ Детектор границ Кэнни записывает результат в одноканальное (полутоновое) изображение

```
#include <opencv2/opencv.hpp>

int main( int argc, char** argv ) {

  cv::Mat img_rgb, img_gry, img_cny;

  cv::namedWindow( "Example Gray", cv::WINDOW_AUTOSIZE );
  cv::namedWindow( "Example Canny", cv::WINDOW_AUTOSIZE );

  img_rgb = cv::imread( argv[1] );

  cv::cvtColor( img_rgb, img_gry, cv::COLOR_BGR2GRAY);
  cv::imshow( "Example Gray", img_gry );

  cv::Canny( img_gry, img_cny, 10, 100, 3, true );
  cv::imshow( "Example Canny", img_cny );

  cv::waitKey(0);

}
```

Это позволяет легко сцеплять операторы. Например, если мы хотим уменьшить изображение дважды, а затем поискать в нем прямые линии, то можем поступить, как показано в примере 2.8.

Пример 2.8 ❖ Комбинирование оператора построения уменьшающейся пирамиды (два раза) и оператора Кэнни в одном конвейере обработки изображения

```
cv::cvtColor( img_rgb, img_gry, cv::BGR2GRAY );
cv::pyrDown( img_gry, img_pyr );
cv::pyrDown( img_pyr, img_pyr2 );
cv::Canny( img_pyr2, img_cny, 10, 100, 3, true );
// сделать что-то с 'img_cny'
//
...
```

В примере 2.9 показан простой способ читать и записывать пиксели изображения из примера 2.8.

Пример 2.9 ❖ Чтение и установка пикселей в примере 2.8

```
int x = 16, y = 32;
cv::Vec3b intensity = img_rgb.at< cv::Vec3b >(y, x);

// ( Примечание: можно было бы написать img_rgb.at< cv::Vec3b >(y, x)[0] )
```

```
//
uchar blue  = intensity[0];
uchar green = intensity[1];
uchar red   = intensity[2];

std::cout << "В позиции (x,y) = (" << x << ", " << y <<
          "): (blue, green, red) = (" <<
          (unsigned int)blue <<
          ", " << (unsigned int)green << ", " <<
          (unsigned int)red << ")" << std::endl;

std::cout << "Полутоновый пиксель: " <<
             (unsigned int)img_gry.at<uchar>(y, x) << std::endl;

x /= 4; y /= 4;

std::cout << "Пиксель pyramid2: " <<
             (unsigned int)img_pyr2.at<uchar>(y, x) << std::endl;

img_cny.at<uchar>(x, y) = 128;         // Установить пиксель в изображении Кэнни равным 128
```

Ввод с камеры

В мире компьютеров под «зрением» понимаются разные вещи. Иногда мы анализируем загруженные откуда-то фотографии. В других случаях анализируется видео, считываемое с диска. А бывает и так, что нужно обрабатывать поток данных, поступающих с какого-то устройства в реальном времени.

OpenCV, а точнее ее часть – HighGUI – предоставляет простые средства для таких ситуаций. Метод аналогичен чтению видео с диска, поскольку объект типа `cv::VideoCapture` одинаково работает что для дискового файла, что для камеры. В первом случае мы задаем путь к файлу, а во втором – идентификатор камеры (обычно `0`, если к системе подключена только одна камера). Значение по умолчанию, равное `-1`, означает «возьми какую-нибудь», оно вполне годится, когда есть только одна камера и выбирать не из чего (дополнительные детали см. в главе 8). Захват видео из файла или с камеры демонстрируется в примере 2.10.

Пример 2.10 ❖ Один и тот же объект может загружать видео с камеры или из файла

```
#include <opencv2/opencv.hpp>
#include <iostream>

int main( int argc, char** argv ) {

  cv::namedWindow( "Example2_10", cv::WINDOW_AUTOSIZE );

  cv::VideoCapture cap;
  if (argc==1) {
    cap.open(0);                       // открыть первую камеру
  } else {
    cap.open(argv[1]);
  }
  if( !cap.isOpened() ) {              // ошибки были?
    std::cerr << "Ошибка при открытии устройства захвата." << std::endl;
    return -1;
  }

  // Дальше, как в примере 2.3
  ...
```

Если задано имя файла, то OpenCV открывает файл, как в примере 2.3, в противном случае пытается открыть камеру с идентификатором `0`. Мы проверяем, было ли что-то открыто; если нет, выводится сообщение об ошибке.

Запись в AVI-файл

Во многих приложениях нужно записывать поступающие потоком входные данные или даже разрозненные изображения в выходной видеопоток, и в OpenCV есть простой способ решения этой задачи. Выше мы создавали устройство захвата, позволяющее получать кадры из видеопотока по одному, но точно так же можно создать записывающее устройство, которое позволит помещать кадры в видеофайл. Для этого предназначен объект типа `cv::VideoWriter`.

Создав такой объект, мы можем потоком передавать ему кадры и в конце вызвать метод `cv::VideoWriter.release()`. Для пущего интереса в примере 2.11 приведена программа, которая открывает видеофайл, читает его содержимое, преобразует его в лог-полярные координаты (то, что реально видит глаз, см. главу 11) и выводит результат в новый файл.

Пример 2.11 ❖ Полная программа чтения цветного видео и записи в лог-полярных координатах

```
#include <opencv2/opencv.hpp>
#include <iostream>

int main( int argc, char* argv[] ) {

    cv::namedWindow( "Example2_11", cv::WINDOW_AUTOSIZE );
    cv::namedWindow( "Log_Polar",   cv::WINDOW_AUTOSIZE );

    // ( Примечание: можно было бы захватывать данные с камеры,
    // если передать ее идентификатор в виде int.)
    //
    cv::VideoCapture capture( argv[1] );

    double fps = capture.get( cv::CAP_PROP_FPS );
    cv::Size size(
      (int)capture.get( cv::CAP_PROP_FRAME_WIDTH ),
      (int)capture.get( cv::CAP_PROP_FRAME_HEIGHT )
    );

    cv::VideoWriter writer;
    writer.open( argv[2], CV_FOURCC('M','J','P','G'), fps, size );

    cv::Mat logpolar_frame, bgr_frame;
    for(;;) {

      capture >> bgr_frame;
      if( bgr_frame.empty() ) break;   // выходим, если больше ничего нет

      cv::imshow( "Example2_11", bgr_frame );

      cv::logPolar(
        bgr_frame,                     // Входной цветной кадр
        logpolar_frame,                // Выходной кадр в лог-полярных координатах
        cv::Point2f(                   // Центр лог-полярного преобразования
          bgr_frame.cols/2,            // x
```

```
            bgr_frame.rows/2            // y
          ),
          40,                           // Абсолютная величина (масштабный параметр)
          cv::WARP_FILL_OUTLIERS        // Заполнить выбросы нулями
        );

        cv::imshow( "Log_Polar", logpolar_frame );
        writer << logpolar_frame;

        char c = cv::waitKey(10);
        if( c == 27 ) break;            // дать пользователю возможность выйти
      }

      capture.release();
}
```

В этой программе много уже знакомых элементов. Мы открываем видео и читаем свойства (число кадров в секунду, ширину и высоту изображения), необходимые, чтобы открыть файл для объекта `cv::VideoWriter`. Затем мы читаем кадры видео с помощью объекта `cv::VideoReader`, преобразуем их в лог-полярные координаты и записываем в новый файл, пока не закончатся исходные данные или пользователь не нажмет клавишу Esc. Затем программа завершается.

Методу `open` объекта `cv::VideoWriter` передается несколько параметров. Первый – имя нового файла. Второй – *видеокодек*, отвечающий за сжатие видеопотока. На рынке полно таких кодеков, но выбирать можно только из тех, что установлены на вашей машине (кодеки устанавливаются отдельно от OpenCV). Мы выбрали сравнительно популярный кодек MJPG и сообщили об этом OpenCV с помощью макроса `CV_FOURCC()`, который в качестве аргументов принимает четыре символа. Эти символы составляют «четырехзначный код», имеющийся у любого кодека. Четырехзначный код кодека *движущийся jpeg* – «MJPG», отсюда и обращение `CV_FOURCC('M','J','P','G')`. Следующие два аргумента – частота воспроизведения кадров и размер изображений. Мы взяли значения, полученные из исходного (цветного) видео.

Резюме

Прежде чем переходить к следующей главе, давайте поймем, где находимся и куда направляемся. Мы видели, что OpenCV предоставляет разнообразные простые в использовании средства для чтения и записи фотографий и видео, а также захвата видео с камеры. Мы также видели, что в библиотеке есть примитивные функции для обработки таких изображений. Но вот чего мы пока не видели, так это более мощных элементов библиотеки, которые позволяют выполнять сложные манипуляции со всем набором абстрактных типов данных, важных для решения практических задач компьютерного зрения.

В следующих двух главах мы более глубоко изучим основные элементы и лучше разберемся как в функциях, относящихся к интерфейсу, так и в типах данных изображений. Мы продолжим изучение примитивных операторов обработки изображений и познакомимся с некоторыми более продвинутыми. Это подготовит нас к исследованию специализированных сервисов для решения таких разнородных задач, как калибровка камеры, распознавание и сопровождение объектов. Готовы? Вперед!

Упражнения

Скачайте и установите OpenCV, если еще не сделали этого. Изучите структуру каталогов. Особое внимание обратите на каталог *docs*, в котором имеется файл *index.htm*, содержащий ссылки на основные разделы документации. Исследуйте также основные части библиотеки. В модуле `core` находятся базовые структуры данных и алгоритмы. Модуль `imgproc` содержит алгоритмы обработки изображений и компьютерного зрения, модуль `ml` – алгоритмы машинного обучения и кластеризации, а модуль `highgui` – функции ввода-вывода. Загляните в каталог *.../samples/cpp*, там много полезных примеров.

1. Следуя инструкциям по установке и сборке, приведенным в этой книге или на сайте http://opencv.org, соберите выпускную и отладочную версии библиотеки. Это может занять некоторое время, но результирующие DLL-файлы нам понадобятся. Не забудьте сказать *cmake* о необходимости собрать примеры в каталоге *.../opencv/samples/*.
2. Перейдите в каталог, куда помещены результаты сборки *.../opencv/samples/* (мы помещали их в каталог *.../trunk/eclipse_build/bin*), и найдите файл *lkdemo.cpp* (пример прослеживания движения). Подключите к своей системе камеру и запустите программу. Выбрав окно отображения, нажмите **r**, чтобы инициализировать прослеживание. Можете добавить точки, щелкая мышью в разных местах видео. Можете переключиться в режим наблюдения за точками (а не за изображением), нажав **n**. Повторное нажатие **n** переключает из режима «день» в режим «ночь» и обратно.
3. Объединив код захвата и сохранения из примера 2.11 с кодом из примера 2.6, напишите программу, которая читает видео с камеры и сохраняет уменьшенные цветные кадры на диск.
4. Объедините программу из упражнения 3 с кодом отображения в окне из примера 2.2, так чтобы обработанные кадры выводились на экран.
5. Объедините программу из упражнения 4 с кодом управления ползунком из примера 2.4, так чтобы пользователь мог динамически изменять коэффициент субдискретизации от 2 до 8. На диск можно не записывать, но выводить результаты на экран необходимо.