



Нижегородский государственный университет им. Н.И. Лобачевского

***Разработка мультимедийных приложений
с использованием библиотек OpenCV и IPP***

Лабораторная работа
Базовые операции обработки изображений

При поддержке компании Intel

Кустикова В.Д.,
кафедра математического обеспечения ЭВМ

Содержание

- ❑ Цели и задачи работы
- ❑ Обзор возможностей модуля `imgproc` библиотеки OpenCV:
 - Свертка и линейные фильтры
 - Сглаживание изображений
 - Морфологические преобразования (эрозия, дилатация и более сложные операции)
 - Операторы Собеля и Лапласа
 - Детектор ребер Канни
 - Вычисление и выравнивание гистограмм
- ❑ Разработка консольного редактора изображений
- ❑ *Разработка редактора изображений с графическим интерфейсом на базе OpenCV (новые реализации на Qt)



Где используется обработка изображений? (1)

- ❑ Основной круг задач компьютерного зрения лежит в сфере **качественного и количественного анализа изображений и потоков видеоданных** (поиск и классификация объектов, сопровождение объектов на видео и др.).
- ❑ Большинство методов решения указанных задач тем или образом использует различные подходы для выполнения предобработки изображений.
- ❑ **Предобработка** подразумевает преобразование исходного изображения в некоторое новое изображение.

Где используется обработка изображений? (2)

□ Примеры практических задач:

- Анализ **медицинских изображений** (МРТ).
- **Классификация изображений** по контексту и организации последующего поиска.
- **Задачи видеоаналитики** (ADAS).
- **Качественный анализ сцен** и распознавание конкретных классов объектов при разработке систем машинного зрения в робототехнике.
- Задачи **повышения качества и контраста снимков**, полученных со спутниковых видеокамер.

Цели работы

- Изучить базовые операции обработки изображений с использованием реализаций соответствующих функций библиотеки компьютерного зрения OpenCV.

Задачи работы (1)

- Изучить принцип работы базовых операций обработки изображений:
 - линейная фильтрация;
 - сглаживание с различными ядрами;
 - морфологические преобразования;
 - применение оператора Собеля;
 - применение оператора Лапласа;
 - определение ребер посредством детектора Канни;
 - вычисление гистограммы;
 - выравнивание гистограммы.

Задачи работы (2)

- ❑ Рассмотреть прототипы функций, реализующих перечисленные операции в библиотеке OpenCV.
- ❑ Разработать простые примеры использования указанного набора функций.
- ❑ Разработать консольный редактор изображений, поддерживающий все представленные операции.
- ❑ Разработать графический редактор с интерфейсом на базе библиотеки OpenCV (реализации компонент на базе Qt).

Тестовая инфраструктура

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

ОБЗОР ВОЗМОЖНОСТЕЙ МОДУЛЯ IMGPROC БИБЛИОТЕКИ OPENCV



Свертка и линейные фильтры

- I – полутоновое изображение
- Линейный фильтр определяется вещественнозначной функцией F , заданной на растре. Данная функция называется **ядром фильтра**, а операция фильтрации выполняется посредством вычисления **дискретной свертки**:

$$I'(x, y) = \sum_i \sum_j F(i, j) \cdot I(x + i, y + j)$$

- Окрестность называется **шаблоном** или **апертурой**.
- Шаблон накладывается на каждый текущий пиксель посредством совмещения пикселя с конкретной точкой шаблона – **ведущей позицией шаблона**.

Свертка и линейные фильтры. Проблемы

- ❑ Текущий пиксель находится на границе изображения?
- ❑ Возможные решения:
 - Обрезать края
 - Не учитывать в процессе суммирования пиксель, который реально не существует
 - Доопределить окрестности граничных пикселей посредством экстраполяции (например, простым дублированием граничных пикселей)
 - Доопределить окрестности граничных пикселей посредством зеркального отражения – завернуть изображение в тор



Линейные фильтры. Функции OpenCV

```
void filter2D(const Mat& src, Mat& dst, int ddepth,  
             const Mat& kernel,  
             Point anchor=Point(-1, -1), double delta=0,  
             int borderType=BORDER_DEFAULT)
```

- ❑ Новое значение интенсивности пикселя вычисляется по формуле:

$$dst(x, y) = \sum_{\substack{0 \leq x' < anchor.x \\ 0 \leq y' < anchor.y}} kernel(x', y') \cdot src(x + x' - anchor.x, y + y' - anchor.y)$$

- ❑ В случае многоканального изображения ядро применяется к каждому каналу в отдельности.



Пример использования

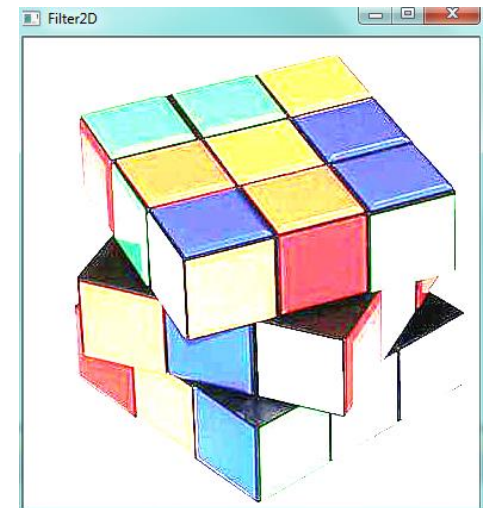
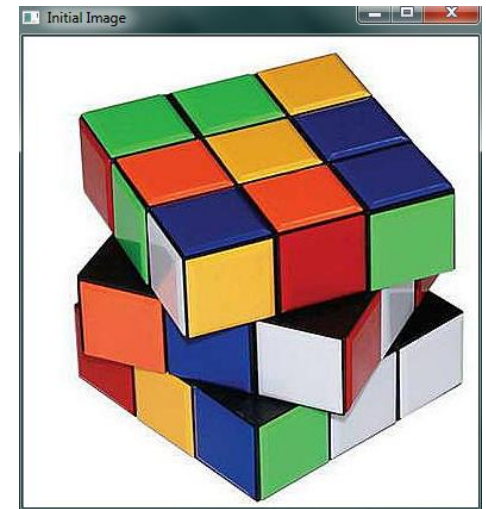
```
const char *initialWinName = "Initial Image",
          *resultWinName = "Filter2D";
const float kernelData[] = {-0.1f, 0.2f, -0.1f,
                             0.2f, 3.0f, 0.2f,
                             -0.1f, 0.2f, -0.1f};

const Mat kernel(3, 3, CV_32FC1,
                 (float *)kernelData);

Mat src, dst;
src = imread(argv[1], 1);
filter2D(src, dst, -1, kernel);

namedWindow(initialWinName, CV_WINDOW_AUTOSIZE);
imshow(initialWinName, src);
namedWindow(resultWinName, CV_WINDOW_AUTOSIZE);
imshow(resultWinName, dst);
waitKey();

destroyAllWindows();
src.release();
dst.release();
```



Сглаживание изображений

- ❑ **Сглаживание** – свертка с ядрами специального вида.
- ❑ Сглаживание играет важную роль при необходимости уменьшить разрешение изображения и получить пирамиду изображений разного масштаба (image pyramids).

Сглаживание изображений. Функции OpenCV (1)

```
void blur(const Mat& src, Mat& dst, Size ksize,  
          Point anchor=Point(-1, -1),  
          int borderType=BORDER_DEFAULT)
```

- Функция **blur** выполняет размытие посредством вычисления свертки исходного изображения с ядром K :

$$K = \frac{1}{kSize.width \cdot kSize.height} \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}$$

Сглаживание изображений. Функции OpenCV (2)

```
void boxFilter(const Mat& src, Mat& dst, int ddepth,  
              Size ksize, Point anchor=Point(-1, -1),  
              bool normalize=true,  
              int borderType=BORDER_DEFAULT)
```

□ Функция **boxFilter** использует ядро более общего вида:

$$K = \alpha \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix}, \text{ где}$$
$$\alpha = \begin{cases} \frac{1}{kSize.width \cdot kSize.height}, & \text{normalize} = \text{true} \\ 1, & \text{в противном случае} \end{cases}$$

Сглаживание изображений. Функции OpenCV (3)

```
void GaussianBlur(const Mat& src, Mat& dst, Size ksize,  
                  double sigmaX, double sigmaY=0,  
                  int borderType=BORDER_DEFAULT)
```

- ❑ Размытия с помощью вычисления свертки изображения с дискретным ядром Гаусса со стандартными отклонениями, равными **sigmaX** и **sigmaY** по осям Oх и Oу соответственно.
- ❑ **Замечание:** ширина и высота ядра (**kSize**) должны быть положительными и нечетными, либо нулевыми, если размер ядра определяется из стандартных отклонений.



Сглаживание изображений. Функции OpenCV (4)

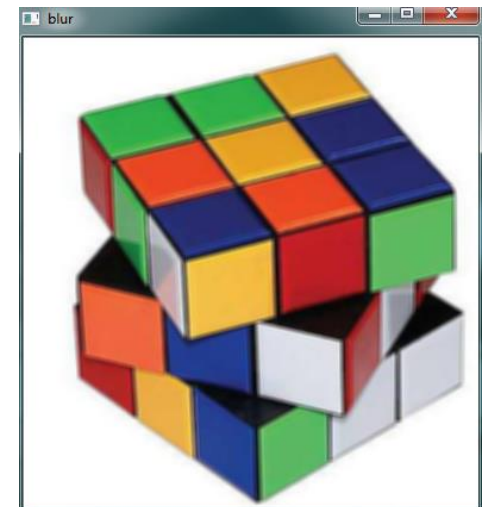
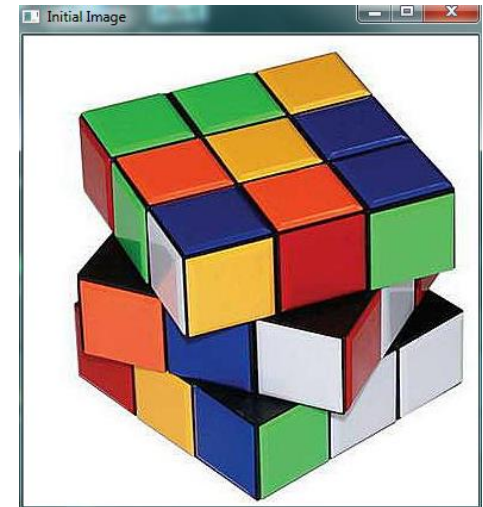
```
void medianBlur(const Mat& src, Mat& dst, int ksize)
```

- ❑ Размытие посредством применения **медианного фильтра**.
- ❑ Медианный фильтр строится подобно линейному фильтру:
 - Выбирается некоторый шаблон.
 - Набор интенсивностей пикселей, которые накрыты шаблоном, сортируются.
 - Выбирается интенсивность, находящаяся в середине отсортированного множества.
- ❑ Медианный фильтр применяется к каждому каналу.



Пример использования

```
const char *initialWinName = "Initial Image",  
          *blurWinName = "blur";  
Mat img, blurImg;  
img = imread(argv[1], 1);  
  
blur(img, blurImg, Size(5, 5));  
  
namedWindow(initialWinName, CV_WINDOW_AUTOSIZE);  
namedWindow(blurWinName, CV_WINDOW_AUTOSIZE);  
imshow(initialWinName, img);  
imshow(blurWinName, blurImg);  
waitKey();  
  
destroyAllWindows();  
  
img.release();  
blurImg.release();
```



Морфологические преобразования.

Дилатация

- ❑ **Дилатация** (морфологическое расширение) – «свертка» изображения или выделенной области изображения с некоторым ядром.
- ❑ Ядро может иметь произвольную форму и размер (во квадрат или круг).
- ❑ При этом в ядре выделяется единственная **ведущая позиция** (anchor), которая совмещается с текущим пикселем при вычислении свертки.
- ❑ Проход шаблоном по изображению и применение **оператора локального максимума к интенсивностям** пикселей изображения, которые накрываются шаблоном => **рост светлых областей**.

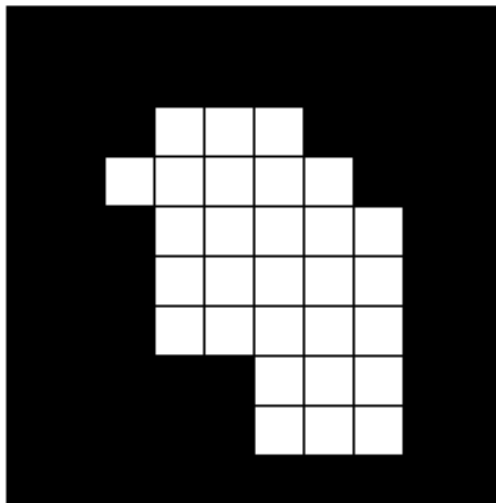


Морфологические преобразования.

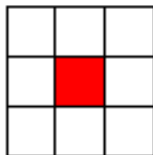
Эрозия

- ❑ **Эрозия** (морфологическое сужение) – операция, обратная к дилатации.
- ❑ Действие эрозии подобно дилатации, разница лишь в том, что используется **оператор поиска локального минимума** => **рост темных областей**.

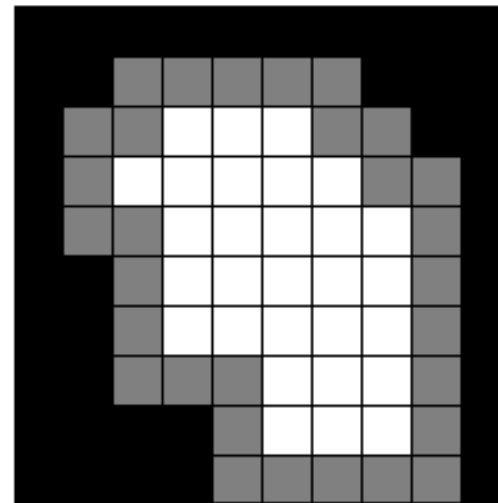
Пример действия дилатации и эрозии



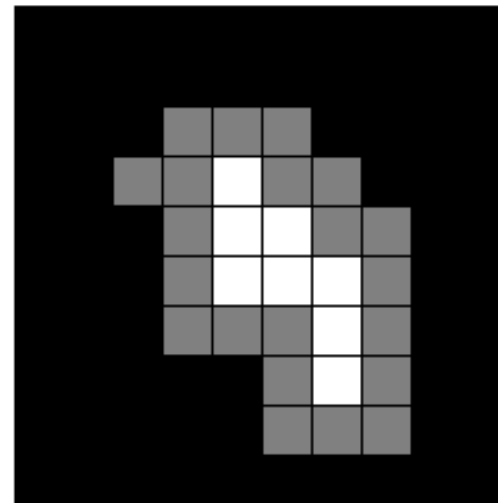
а) исходное изображение



б) шаблон (центр – ведущий элемент)



с) результат дилатации



д) результат эрозии

Дилатация и эрозия. Функции OpenCV

```
void dilate(const Mat& src, Mat& dst, const Mat& element,  
            Point anchor=Point(-1, -1), int iterations=1,  
            int borderType=BORDER_CONSTANT,  
            const Scalar& borderValue =  
                morphologyDefaultBorderValue())
```

```
void erode(const Mat& src, Mat& dst, const Mat& element,  
            Point anchor=Point(-1, -1), int iterations=1,  
            int borderType=BORDER_CONSTANT,  
            const Scalar& borderValue =  
                morphologyDefaultBorderValue())
```

- ❑ **element** – шаблон, который используется в процессе дилатации. Если **element=Mat()**, то применяется квадратный шаблон размером 3x3.

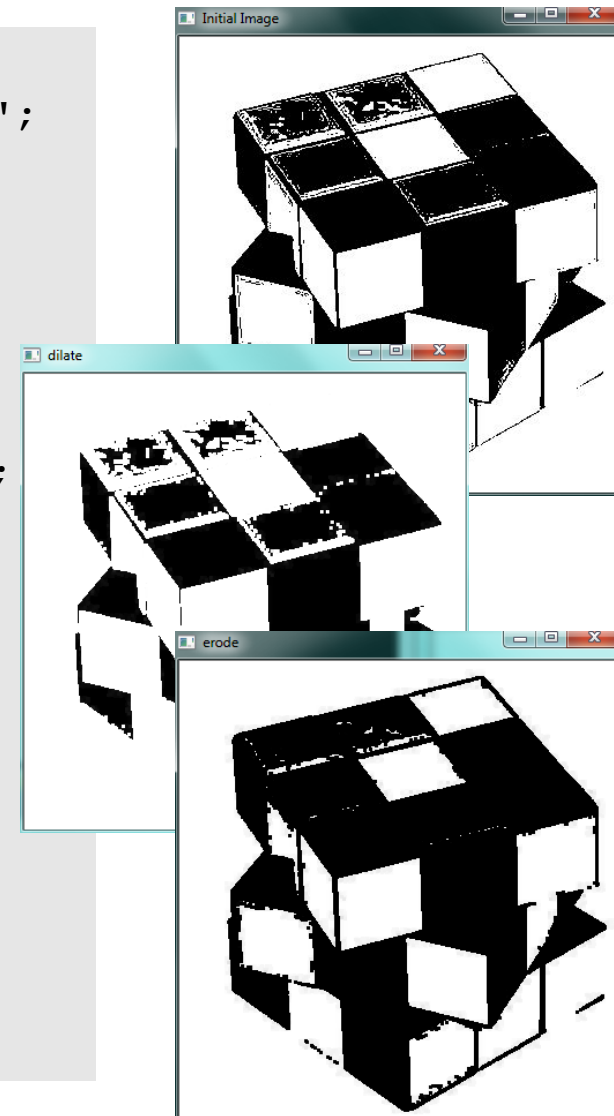


Пример использования

```
const char *initialWinName="Initial Image",
    *erodeWinName="erode", *dilateWinName="dilate";
Mat img, erodeImg, dilateImg, element;
img = imread(argv[1], 1);
element = Mat();
erode(img, erodeImg, element);
dilate(img, dilateImg, element);

namedWindow(initialWinName, CV_WINDOW_AUTOSIZE);
namedWindow(erodeWinName, CV_WINDOW_AUTOSIZE);
namedWindow(dilateWinName, CV_WINDOW_AUTOSIZE);
imshow(initialWinName, img);
imshow(erodeWinName, erodeImg);
imshow(dilateWinName, dilateImg);
waitKey();

// освобождение ресурсов
img.release();
erodeImg.release();
dilateImg.release();
```



Дополнительные морфологические операции

- ❑ **Размыкание** – дилатация от эрозии. Позволяет удалить все мелкие объекты и шум на изображении.
- ❑ **Замыкание** – эрозия от дилатации. Позволяет удалить небольшие внутренние «дырки» и убрать зернистость по краям области.
- ❑ **Морфологический градиент** – разница дилатации и эрозии. Морфологический градиент обеспечивает поиск контуров объектов.
- ❑ **«Верх шляпы»** – разница исходного изображения и результата размыкания.
- ❑ **«Черная шляпа»** – разница результата замыкания и исходного изображения.



Дополнительные морфологические операции. Функции OpenCV

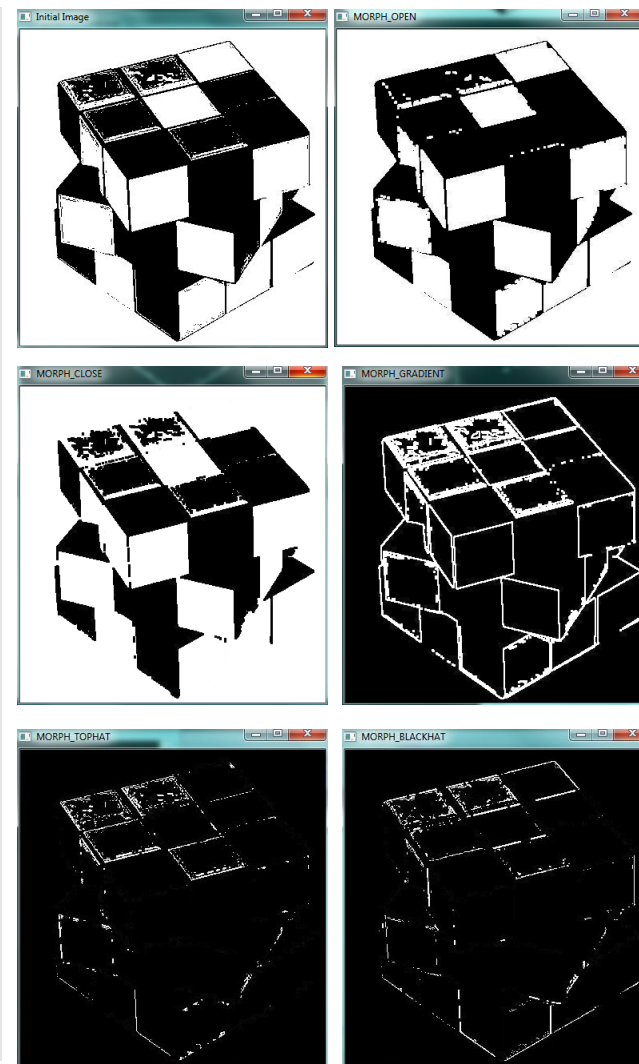
```
void morphologyEx(const Mat& src, Mat& dst, int op,  
                 const Mat& element,  
                 Point anchor=Point(-1, -1),  
                 int iterations=1,  
                 int borderType=BORDER_CONSTANT,  
                 const Scalar& borderValue =  
                     morphologyDefaultBorderValue())
```

- **op** – тип морфологической операции:
 - **MORPH_OPEN** – размыкание
 - **MORPH_CLOSE** – замыкание
 - **MORPH_GRADIENT** – морфологический градиент
 - **MORPH_TOPHAT** – «верх шляпы»
 - **MORPH_BLACKHAT** – «черная шляпа»



Дополнительные морфологические операции. Пример использования

```
Mat img,morphologyOpenImg,morphologyCloseImg,  
    morphologyGradientImg,morphologyTopHatImg,  
    morphologyBlackHatImg,element;  
// загрузка изображения  
img = imread(argv[1], 1);  
// применение морфологических операций  
element = Mat();  
morphologyEx(img, morphologyOpenImg,  
             MORPH_OPEN, element);  
morphologyEx(img, morphologyCloseImg,  
             MORPH_CLOSE, element);  
morphologyEx(img, morphologyGradientImg,  
             MORPH_GRADIENT, element);  
morphologyEx(img, morphologyTopHatImg,  
             MORPH_TOPHAT, element);  
morphologyEx(img, morphologyBlackHatImg,  
             MORPH_BLACKHAT, element);  
  
// отображение результата  
// и освобождение памяти...
```



Оператор Собеля (1)

- ❑ **Оператор Собеля** – дискретный дифференциальный оператор, вычисляющий приближенные значения производных разного порядка для функции яркости пикселей.
- ❑ Используется для определения краев/границ посредством приближенного вычисления градиента функции интенсивности.

Оператор Собеля (2)

- Вычисление свертки исходного изображения с ядрами G_x и G_y , обеспечивающими вычисление первых производных по направлениям:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

- Магнитуда градиента: $|G^{ij}| = \sqrt{(G_x^{ij})^2 + (G_y^{ij})^2}$.
- Направление градиента: $\theta^{ij} = \arctan\left(\frac{G_y}{G_x}\right)$.

Оператор Собеля. Функции OpenCV

```
void Sobel(const Mat& src, Mat& dst, int ddepth,  
           int xorder, int yorder, int ksize=3,  
           double scale=1, double delta=0,  
           int borderType=BORDER_DEFAULT)
```

- ❑ **xorder** – порядок производной по оси Oх.
- ❑ **yorder** – порядок производной по оси Oy.
- ❑ **ksize** – размер расширенного ядра оператора Собеля. Принимает одно из значений 1, 3, 5 или 7. Во всех случаях ядро имеет размер **kSize** x **kSize**, кроме ситуации, когда **kSize=1**. При **kSize=1** ядра имеют размер 3x1 или 1x3, по существу применяется фильтр Гаусса. По умолчанию ядро имеет размер 3x3.



Пример использования

```
const char *initialWinName = "Initial Image",
    *xGradWinName = "Gradient in the direction Ox",
    *yGradWinName = "Gradient in the direction Oy",
    *gradWinName = "Gradient";
int ddepth = CV_16S;
double alpha = 0.5, beta = 0.5;
Mat img, grayImg, xGrad, yGrad,
    xGradAbs, yGradAbs, grad;
img = imread(argv[1], 1);
GaussianBlur(img, img, Size(3,3),
    0, 0, BORDER_DEFAULT);
cvtColor(img, grayImg, CV_RGB2GRAY);
Sobel(grayImg, xGrad, ddepth, 1, 0); // по Ox
Sobel(grayImg, yGrad, ddepth, 0, 1); // по Oy
// преобразование градиентов в 8-битные
convertScaleAbs(xGrad, xGradAbs);
convertScaleAbs(yGrad, yGradAbs);
// поэлементное вычисление взвешенной суммы
addWeighted(xGradAbs, alpha, yGradAbs, beta, 0, grad);
// отображение результата и освобождение памяти ...
```



Оператор Лапласа

- ❑ **Оператор Лапласа:** $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$.
- ❑ Дискретный аналог оператора Лапласа используется при обработке изображений.
- ❑ Определение ребер объектов на изображении:
 - Ребра формируются из множества пикселей, в которых оператор Лапласа принимает нулевые значения.
 - Нули вторых производных функции соответствуют экстремальным перепадам интенсивности.

Оператор Лапласа. Функции OpenCV

```
void Laplacian(const Mat& src, Mat& dst, int ddepth,  
              int ksize=1, double scale=1,  
              double delta=0,  
              int borderType=BORDER_DEFAULT)
```

- **kSize** – размер апертуры для вычисление второй производной, является положительным четным числом. При использовании значения по умолчанию **kSize=1** применяется апертура размером 3x3 и ядро представляется матрицей:

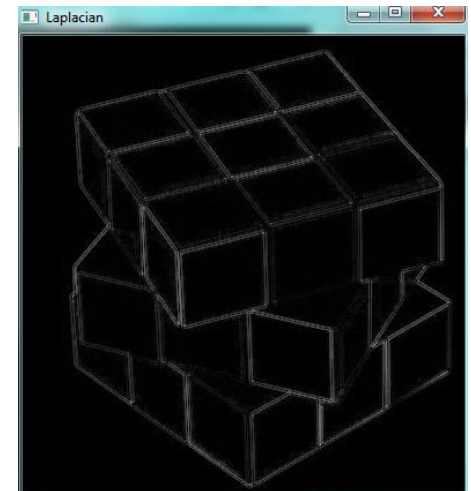
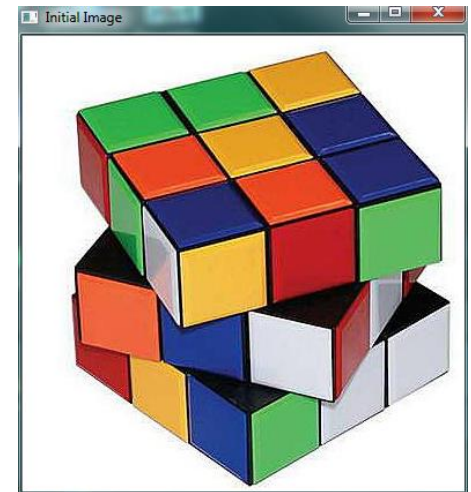
$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Пример использования

```
const char *initialWinName = "Initial Image",
          *laplacianWinName = "Laplacian";
Mat img, grayImg, laplacianImg, laplacianImgAbs;
int ddepth = CV_16S;
img = imread(argv[1], 1);
GaussianBlur(img, img, Size(3,3),
             0, 0, BORDER_DEFAULT);
cvtColor(img, grayImg, CV_RGB2GRAY);
Laplacian(grayImg, laplacianImg, ddepth);
convertScaleAbs(laplacianImg, laplacianImgAbs);

namedWindow(initialWinName, CV_WINDOW_AUTOSIZE);
namedWindow(laplacianWinName, CV_WINDOW_AUTOSIZE);
imshow(initialWinName, img);
imshow(laplacianWinName, laplacianImgAbs);
waitKey();

destroyAllWindows();
img.release(); grayImg.release();
laplacianImg.release(); laplacianImgAbs.release();
```



Детектор Канни (1)

- ❑ Детектор ребер Канни предназначен для поиска границ объектов на изображении.
- ❑ Работа включает несколько этапов:
 - Удаление шума на изображении посредством применения фильтра Гаусса с ядром размера 5:

$$K = \frac{1}{159} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Детектор Канни (2)

- Вычисление первых производных (магнитуд и направлений) посредством применения оператора Собеля с ядрами G_x и G_y .
- Направления градиентов округляются до одного из возможных значений 0^0 , 45^0 , 90^0 , 135^0 .
- Отбор пикселей, которые потенциально принадлежат ребру с использованием процедуры non-maximum suppression. Пиксели, которым соответствуют вектора производных по направлениям, являющиеся локальными максимумами, считаются потенциальными кандидатами на принадлежность ребру.
- Двойное отсечение (гистерезис).

Детектор Канни. Функции OpenCV

```
void Canny(const Mat& image, Mat& edges,  
           double threshold1, double threshold2,  
           int apertureSize=3, bool L2gradient=false)
```

- ❑ **threshold1, threshold2** – параметры алгоритма, пороговые значения для отсеечения.
- ❑ **apertureSize** – размер апертуры для применения оператора Собеля.
- ❑ **L2gradient** – флаг, который указывает, по какой норме будет вычисляться магнитуда градиента. Принимает истинное значение, если используется норма L_2 , в противном случае L_1 .



Пример использования

```
const char *cannyWinName = "Canny detector";
Mat img, grayImg, edgesImg;
double lowThreshold = 70, uppThreshold = 260;
img = imread(argv[1], 1);

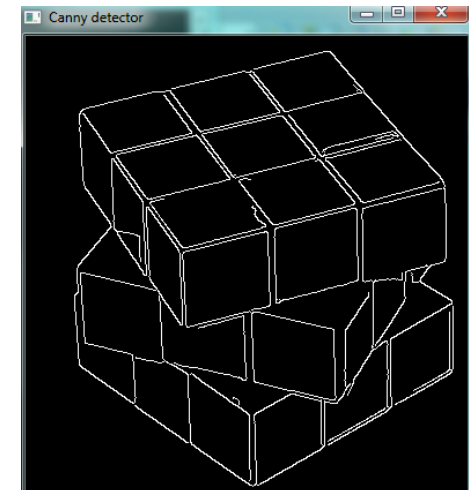
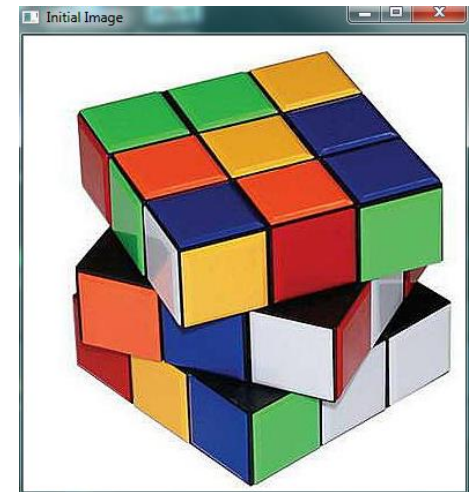
blur(img, img, Size(3,3));
cvtColor(img, grayImg, CV_RGB2GRAY);

Canny(grayImg, edgesImg,
      lowThreshold, uppThreshold);

namedWindow(cannyWinName, CV_WINDOW_AUTOSIZE);
imshow(cannyWinName, edgesImg);
waitKey();

destroyAllWindows();

img.release();
grayImg.release();
edgesImg.release();
```



Вычисление гистограмм (1)

- ❑ Один из наиболее распространенных дефектов фотографических, сканерных и телевизионных изображений – слабый контраст.
- ❑ **Контрастом** – разность максимального и минимального значений яркости.
- ❑ Существует несколько методов повышения контрастности, основанных на вычислении **гистограммы**.



Вычисление гистограмм (2)

- ❑ Допустим, что имеется изображение в оттенках серого, интенсивность пикселей которого изменяется в пределах значений от a до b , где $a \geq 0$ и $b \leq 255$.
- ❑ Для изображения можно построить гистограмму со столбцами, отвечающими количеству пикселей определенной интенсивности.
- ❑ Гистограмма позволяет представить распределение оттенков на изображении – статистическая картина о распределении интенсивностей.

Вычисление гистограмм. Функции OpenCV

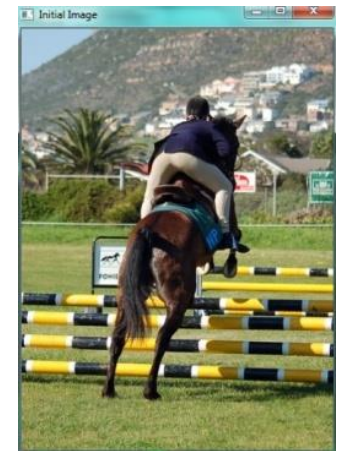
```
void calcHist(const Mat* arrays, int narrays,  
             const int* channels, const Mat& mask,  
             MatND& hist, int dims, const int* histSize,  
             const float** ranges, bool uniform=true,  
             bool accumulate=false)
```

```
void calcHist(const Mat* arrays, int narrays,  
             const int* channels, const Mat& mask,  
             SparseMat& hist, int dims,  
             const int* histSize, const float** ranges,  
             bool uniform=true, bool accumulate=false)
```

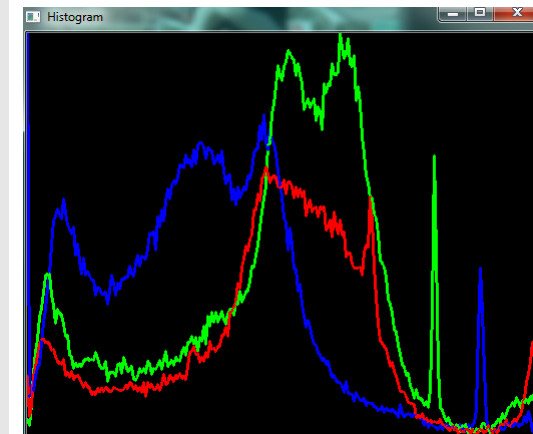


Пример использования

```
Mat img,bgrChannels[3],bHist,gHist,rHist,histImg;
int kBins = 256; // количество бинов гистограммы
float range[] = {0.0f, 256.0f};
const float* histRange = { range };
bool uniform = true;
bool accumulate = false;
int histWidth = 512, histHeight = 400;
int binWidth = cvRound((double)histWidth / kBins);
int i, kChannels = 3;
Scalar colors[] = {Scalar(255, 0, 0),
                  Scalar(0, 255, 0), Scalar(0, 0, 255)};
img = imread(argv[1], 1);
split(img, bgrChannels);
calcHist(&bgrChannels[0], 1, 0, Mat(), bHist, 1,
        &kBins, &histRange, uniform, accumulate);
calcHist(&bgrChannels[1], 1, 0, Mat(), gHist, 1,
        &kBins, &histRange, uniform, accumulate);
calcHist(&bgrChannels[2], 1, 0, Mat(), rHist, 1,
        &kBins, &histRange, uniform, accumulate);
```



Изображение из базы
VOC 2007



Методы повышения контраста изображения

- Методы повышения контраста изображения:
 - линейная растяжка гистограммы (линейное контрастирование),
 - нормализация гистограммы,
 - выравнивание (линеаризация или эквализация, equalization) гистограммы.

Линейная растяжка гистограммы

- Если интенсивности исходного изображения изменялись в диапазоне от i_1 до i_2 , тогда необходимо **линейно «растянуть»** указанный **диапазон** так, чтобы значения изменялись от 0 до 255.
- Необходимо пересчитать старые значения интенсивности f_{xy} для всех пикселей (x, y) :

$$g_{xy} = a \cdot f_{xy} + b,$$

где коэффициенты a, b просто вычисляются, исходя из того, что граница i_1 должна перейти в 0, а i_2 – в 255.

Нормализация гистограммы

- ❑ Обеспечивает **растяжку** наиболее **информативной части**.
- ❑ **Информативная часть** – набор пиков гистограммы, т.е. интенсивности, которые чаще остальных встречаются на изображении.
- ❑ Схема нормализации:
 - Бины, соответствующие редко встречающимся интенсивностям, в процессе нормализации отбрасываются.
 - Выполняется обычная линейная растяжка получившейся гистограммы.

Выравнивание гистограммы (1)

- Цель выравнивания состоит в том, чтобы все уровни яркости имели бы одинаковую частоту, а **гистограмма соответствовала равномерному закону распределения**.
- Обозначения:
 - x, y – случайные величины, описывающие изменение интенсивности пикселей на изображениях;
 - $w_x(x)$ – плотность распределения интенсивности на исходном изображении;
 - $w_y(y)$ – желаемая плотность распределения;
 - $F_x(x)$ и $F_y(y)$ интегральные законы распределения случайных величин x и y .

Выравнивание гистограммы (2)

- Необходимо найти преобразование плотностей распределения $y = f(x)$, которое позволило бы получить желаемую плотность:

$$w_y(y) = \begin{cases} \frac{1}{y_{max} - y_{min}}, & y_{min} \leq y \leq y_{max} \\ 0, & \text{в противном случае} \end{cases}$$

- Из условия вероятностной эквивалентности следует, что $F_x(x) = F_y(y)$. Распишем интегральный закон распределения по определению:

$$F_x(x) = F_y(y) = \int_{y_{min}}^y w_y(y) dy = \frac{y - y_{min}}{y_{max} - y_{min}}$$

Выравнивание гистограммы (3)

- В результате интегрирования получаем:

$$y = (y_{max} - y_{min})F_x(x) + y_{min}$$

- **Как оценить интегральный закон распределения $F_x(x)$?**
- Построить гистограмму исходного изображения.
- Нормализовать полученную гистограмму.
- Значения бинов можно рассматривать как приближенное значение функции плотности распределения $w_x^*(x)$, $0 \leq x \leq 255$. Значение интегральной функции распределения можно представить как сумму следующего вида:

$$F_x^*(x) = \sum_{j=0}^x w_x^*(j)$$

Выравнивание гистограммы. Функции OpenCV

```
void equalizeHist(const Mat& src, Mat& dst)
```

- Функция работает в четыре этапа:
 - Вычисление гистограммы H исходного изображения **src**. Отметим, что **src** – 8-битное одноканальное изображение.
 - Нормализация гистограммы. Нормализация посредством деления величины каждого бина гистограммы на общее количество пикселей.
 - Построение интегральной гистограммы $H'_i = \sum_{0 \leq j < i} H_j$.
 - Определение нового значения интенсивности пикселя $\text{dst}(x,y) = H'(\text{src}(x,y))$.

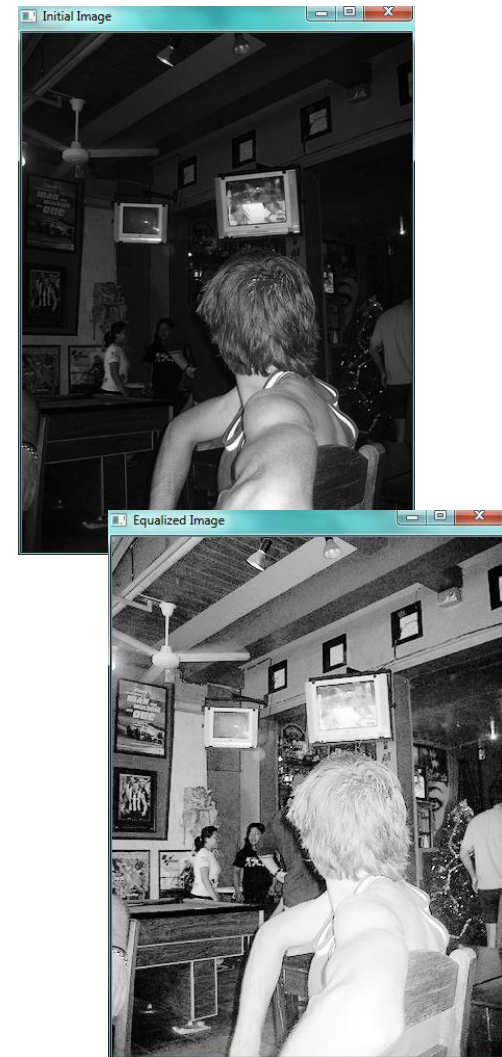


Пример использования

```
const char *initialWinName = "Initial Image",
          *equalizedWinName = "Equalized Image";
Mat img, grayImg, equalizedImg;
img = imread(argv[1], 1);
// преобразование в оттенки серого
cvtColor(img, grayImg, CV_RGB2GRAY);
// выравнивание гистограммы
equalizeHist(grayImg, equalizedImg);

// отображение исходного изображения и гистограмм
namedWindow(initialWinName, CV_WINDOW_AUTOSIZE);
namedWindow(equalizedWinName, CV_WINDOW_AUTOSIZE);
imshow(initialWinName, grayImg);
imshow(equalizedWinName, equalizedImg);
waitKey();

destroyAllWindows();
img.release();
grayImg.release();
equalizedImg.release();
```



РАЗРАБОТКА КОНСОЛЬНОГО РЕДАКТОРА ИЗОБРАЖЕНИЙ



Требования к приложению

- ❑ Организация диалога с пользователем. Предполагается вывод перечня пунктов меню (загрузка изображения и набор операций) и возможность выбора определенной операции.
- ❑ Программа должна обеспечивать многократное выполнение различных операций над разными изображениями.
- ❑ Отображение исходного изображения.
- ❑ Отображение результата применения операции.
- ❑ Контроль корректности вводимых пользователем данных.

Пример диалога

- Пример организации диалога с пользователем:

```
Menu items:
 0 - Read image
 1 - Apply linear filter
 2 - Apply blur<...>
 3 - Apply medianBlur<...>
 4 - Apply GaussianBlur<...>
 5 - Apply erode<...>
 6 - Apply dilate<...>
 7 - Apply Sobel<...>
 8 - Apply Laplacian<...>
 9 - Apply Canny<...>
10 - Apply calcHist<...>
11 - Apply equalizeHist<...>

Input item identifier to apply operation: 0
Input full file name: _
```

Структура приложения (1)

- ❑ **kMenuTabs** – количество пунктов меню (фактически число допустимых операций);

```
const int kMenuTabs = 12;
```

- ❑ **menu** – массив строк, содержащий описание пунктов МЕНЮ:

```
const char* menu[] =  
{  
    "0 - Read image",  
    "1 - Apply linear filter",  
    "2 - Apply blur(...)",  
    ...  
};
```

Структура приложения (2)

- **winNames** – массив строк, содержащий названия окон, которые будут использованы при отображении результата выполнения той или иной операции;

```
const char* winNames[] = {  
    "Initial image",  
    "filter2d",  
    "blur",  
    "medianBlur",  
    ...  
};
```

- **escCode** – код символа ESC (выход из приложения)

```
const int escCode = 27;
```

Основная функция

```
int main(int argc, char** argv) {  
    Mat srcImg;  char ans;  int activeMenuTab = -1;  
    do {  
        // вызов функции выбора пункта меню  
        chooseMenuTab(activeMenuTab, srcImg);  
        // применение операций  
        applyOperation(srcImg, activeMenuTab);  
        printf("Do you want to continue? ESC - exit\n");  
        ans = waitKey();  
    } while (ans != escCode);  
    destroyAllWindows(); // закрытие всех окон  
    srcImg.release(); // освобождение памяти  
    return 0;  
}
```



Вспомогательные функции (1)

- ❑ **printMenu** – функция вывода пунктов меню на консоль.
- ❑ **chooseMenuTab** – функция выбора пункта меню:
 - запрос ввода идентификатора пункта меню;
 - повторный ввод, если введено недопустимое значение.
- ❑ **Замечание:** операции обработки изображения не могут быть выбраны, пока не загружено изображение. Функция возвращает индекс выбранного меню **activeMenuTab** и загруженное изображение **srcImg**.

Вспомогательные функции (2)

- ❑ **loadImage** – функция загрузки изображения:
 - цикл, в котором выполняется запрос на ввод полного имени изображения с контролем корректности операции чтения изображения;
 - возвращает загруженное изображение **srcImg**.

- ❑ **applyOperation** – функция применения операции обработки изображения (в новый модуль):
 - на входе исходное изображение **src** и индекс операции **operationIdx**;
 - содержит оператор множественного выбора;
 - отвечает за отображение изображений.



Решение с набором проектов

- ❑ **02_ImageProcessing.sln** – решение:
 - **01_ConsoleImageEditor** – консольный редактор
 - **02_QtImageEditor** – редактор с графическим интерфейсом
 - **Sample_*** (blur, calcHist, Canny, equalizeHist, erode_dilate, filter2D, Laplacian, morphologyEx, Sobel) – проекты, содержащие примеры использования функций обработки изображений, рассмотренные в работе.



Использование графических компонент библиотеки OpenCV,
реализованных на базе Qt

***РАЗРАБОТКА РЕДАКТОРА С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ**



Подготовка инфраструктуры

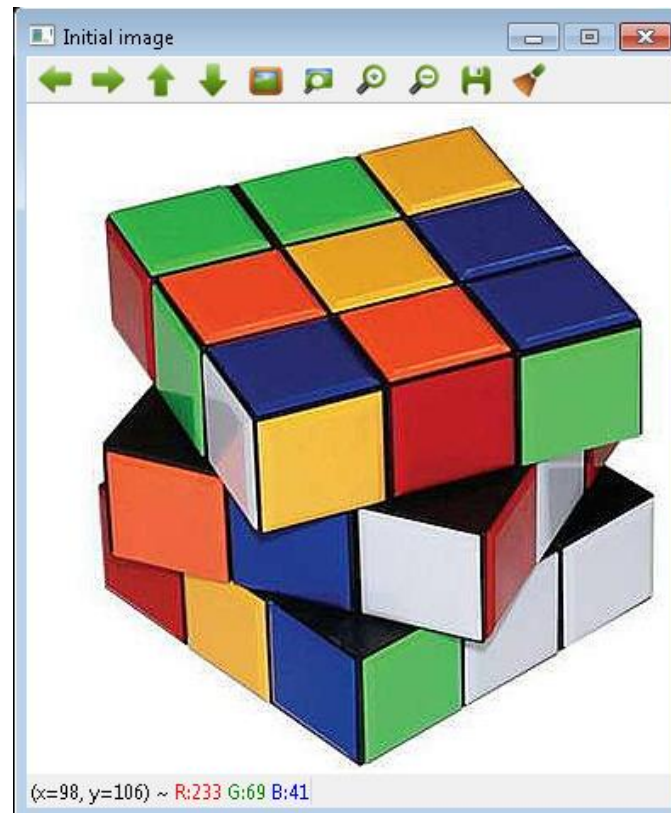
- ❑ Установка библиотек Qt:
 - загрузить Qt libraries 4.x for Windows (VS 2010) с официальной страницы проекта
 - следовать подсказкам инсталлятора.
- ❑ Сборка исходных кодов библиотеки OpenCV с опцией (**-D WITH_QT=YES**).
- ❑ Создание консольного приложения в среде Microsoft Visual Studio и установка свойств проекта, обеспечивающих использование функционала библиотеки OpenCV.



Примеры функций создания графических компонент (1)

□ Создание окна:

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



Примеры функций создания графических компонент (2)

- ❑ Создание кнопочных элементов:

```
createButton(const string& button_name CV_DEFAULT(NULL) ,  
            ButtonCallback on_change CV_DEFAULT(NULL) ,  
            void* userdata CV_DEFAULT(NULL) ,  
            int button_type CV_DEFAULT(CV_PUSH_BUTTON) ,  
            int initial_button_state CV_DEFAULT(0))
```

- ❑ CV_PUSH_BUTTON
- ❑ CV_CHECKBOX
- ❑ CV_RADIOBOX

02_QtImageEditor.exe settings

1 - Apply linear filter 2 - Apply blur(...) 3 - Apply medianBlur(...) 4 - Apply GaussianBlur(...) 5 - Apply erode(...) 6 - Apply dilate(...) 7 - Apply Sobel(...) 8 - Apply Laplacian(...) 9 - Apply Canny(...) 10 - Apply calcHist(...) 11 - Apply equalizeHist(...)



Полный перечень функций

- ❑ Страница документации OpenCV:
 - http://opencv.willowgarage.com/documentation/cpp/highgui_qt_new_functions.html



Постановка задачи

- ❑ Разработать приложение с графическим интерфейсом с использованием функционала библиотеки OpenCV.
- ❑ Приложение должно обеспечивать применение рассмотренных в работе.

Задания для самостоятельной работы

- ❑ Добавьте в разработанную структуру консольного графического редактора поддержку операций, описанных в данной лабораторной работе.
- ❑ Добавьте в разработанный консольный графический редактор возможность сохранения изображений, которые получены в результате применения различных операций.
- ❑ Добавьте возможность рисования геометрических примитивов в разработанный консольный редактор. Предусмотрите возможность удаления отрисованных примитивов.
- ❑ Разработайте редактор изображений с использованием графических компонент библиотеки OpenCV, реализованных на базе Qt.



Авторский коллектив

- ❑ Кустикова Валентина Дмитриевна,
ассистент кафедры
Математического обеспечения ЭВМ факультета ВМК ННГУ
valentina.kustikova@gmail.com

