OpenCV

Przetwarzanie Obrazów

POBR

Najprostszy program

Plik projektu Qt ....

```
QT       += core
QT       -= gui
TARGET = FrameworkQt
CONFIG += console
CONFIG -= app_bundle
TEMPLATE = app
SOURCES += main.cpp
LIBS += -lopencv_core -lopencv_highgui
```

Konfiguracja na Windows #1

- Rozpakowujemy OpenCV do katalogu **C:/Program Files/OpenCV-2.4.10**
- Definiujemy zmienną środowiskową: **OPENCV_DIR** **C:/... /OpenCV-2.4.10/build/x86/vc12**
- Tworzymy projekt typu C++ Win32 Console
- We właściwościach projektu ustawiamy:
  –>Configuration Properties
  —->C/C++
  ——>General
  ——->Additional Include Directories
  ${OPENCV_DIR}/../../include;

Konfiguracja na Windows #2

- We właściwościach projektu ustawiamy:
  –>Configuration Propertis
  —->Linker
  ——>General
  ——->Additional Library Directories
  ${OPENCV_DIR}/lib
- We właściwościach projektu:
  –>Configuration Propertis
  —->Linker
  ——>Additional
  ——-> Dependencies
  dodajemy biblioteki debug:
  opencv_core2410d.lib
  opencv_highgui2410d.lib

Konfiguracja na Windows #3

- W ścieżce PATH dodajemy ścieżkę do katalogu bin z dll: **C:/Program Files/OpenCV-2.4.10/build/x86/vc12/bin**
- Umieszczamy plik **Lena.png** w katalogu z projektem *exe jest w ../Debug, ale katalogiem roboczym przy starcie z Visual Studio jest katalog projektu)

CMake - przenośność konfiguracji Windows/Linux

- Plik **CMakeList.txt**:

```
cmake_minimum_required(VERSION 2.8)
 project ( FmameworkCmake)
find_package( OpenCV REQUIRED)
 include_directories ( ${OpenCV_INCLUDE_DIRS} )
add_executable( FrameworkCmake FrameworkCmake.cpp )
 target_link_libraries ( FrameworkCmake ${OpenCV_LIBS} )
```

- W celu budowy piszemy:

  cmake .

  make

Makefile - wersja Linux #1

```
RM=rm -f
CPPFLAGS=-std=c++11 -pthread
SRCS=main.cpp
# replace all .cpp with .o
OBJS=$(subst .cpp,.o,$(SRCS))
LDLIBS=-lopencv_core -lopencv_highgui
LDFLAGS=
TARGET=Framework

all : $(TARGET)

$(TARGET): $(OBJS)
    g++ $(LDFLAGS) -o $(TARGET) $(OBJS) $(LDLIBS)
```

Makefile - wersja Linux #2

```
depend: .depend
.depend: $(SRCS)
	rm -f ./.depend
	$(CXX) $(CPPFLAGS) -MM $^>>./.depend;
include .depend
clean:
	$(RM) $(OBJS)
dist-clean: clean
	$(RM) *~ .dependtool
	$(RM) $(TARGET)
Release:$(TARGET)
Debug:$(TARGET)
cleanDebug:dist-clean
cleanRelease: dist-clean
```

Najprostszy program

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
int main(int, char *[]) {
    cv::Mat image = cv::imread("Lena.png");
    cv::namedWindow("Lena");
    cv::imshow("Lena",image);
    cv::waitKey(-1);
    return 0;
}
```

Framework ... funkcja main

```
int main(int, char *[]) {
  std::cout << "Start ..." << std::endl;
  cv::Mat image = cv::imread("Lena.png");
  cv::namedWindow("Lena");
  //cv::Mat image2 = image(cv::Range(100,200),cv::Range(100,200));
  cv::Mat image2 = image(cv::Rect(100,100,100,100));
  perform(image2);
  cv::Mat max = selectMax(image);
  cv::imshow("Lena",image);
  cv::imshow("Max",max);
  std::cout << image2.isContinuous()<<max.isContinuous()<<std::endl;
  cv::waitKey(-1);
  return 0;
}
```

Framework ... funkcja selectMax

```
cv::Mat selectMax(cv::Mat& I){
    cv::Mat res(I.rows,I.cols, CV_8UC3);
    switch(I.channels())  {
    case 3:
        cv::Mat_<cv::Vec3b> _I = I;
        cv::Mat_<cv::Vec3b> _R = res;
        for( int i = 0; i < I.rows; ++i)
            for( int j = 0; j < I.cols; ++j ){
                int sel = (_I(i,j)[0] < _I(i,j)[1])?1:0;
                sel = _I(i,j)[sel] < _I(i,j)[2]?2: sel;
                _R(i,j)[0] = sel==0?255:0;
                _R(i,j)[1] = sel==1?255:0;
                _R(i,j)[2] = sel==2?255:0;
            }
        res = _R;
        break;
    }
    return res;
}
```

Framework ... funkcja perform

```
cv::Mat& perform(cv::Mat& I){
  CV_Assert(I.depth() != sizeof(uchar));
  switch(I.channels())  {
  case 1:
    for( int i = 0; i < I.rows; ++i)
        for( int j = 0; j < I.cols; ++j )
            I.at<uchar>(i,j) = (I.at<uchar>(i,j)/32)*32;
    break;
     ...
  }
  return I;
}
```

Framework ... funkcja perform

```
cv::Mat& perform(cv::Mat& I){
  CV_Assert(I.depth() != sizeof(uchar));
  switch(I.channels())  {
     ...
  case 3:
    cv::MatIterator_<cv::Vec3b> it, end;
    for(  it  = I.begin<cv::Vec3b>(), end = I.end<cv::Vec3b>(); it != end; +
        (*it)[0]  = ((*it)[0]/32)*32;
        (*it)[1]  = ((*it)[1]/32)*32;
        (*it)[2]  = ((*it)[2]/32)*32;
    }
    break;
  }
  return I;
}
```

Framework ... funkcja perform

```
cv::Mat& perform(cv::Mat& I){
  CV_Assert(I.depth() != sizeof(uchar));
  switch(I.channels())  {
     ...
  case 3:
    cv::Mat_<cv::Vec3b> _I = I;
    for( int i = 0; i < I.rows; ++i)
         for( int j = 0; j < I.cols; ++j ){
             _I(i,j)[0] = (_I(i,j)[0]/32)*32;
             _I(i,j)[1] = (_I(i,j)[1]/32)*32;
             _I(i,j)[2] = (_I(i,j)[2]/32)*32;
         }
    I = _I;
    break;
  }
  return I;
}
```

Mat - operacje na składowych

Rozdzielenie na składowe:

```
vector<Mat> planes;
split (src, planes);
```

Złączenie:

```
vector<Mat> planes;
merge(planes,dst);
```

Operacje na całych planach

```
vector<Mat> planes;
split (src, planes);
planes[0]=planes[1]+2*planes[2];
```

Operacje na fragmentach

```
Rect roi(10, 20, 100, 50);
Mat image_roi = image(roi);
```

Mat - kolejkowanie wywołań

Jedna funkcja może obsługiwać różne strategie

header:

```
cv::Mat polar(const cv::Mat src, cv::Mat dst = cv::Mat());
```

implementacja:

```
cv::Mat polar(const cv::Mat src, cv::Mat dst) {
    dst.create(src.size(), src.type()); // jezeli rozmiar lub typ nie pasuje
....
    return dst ;
}
```

wykorzystanie:

```
Mat p_fft = polar(m_fft); // alokuj
...
Mat p_fft = Mat(m_fft.size(),m_fft.type());
polar(m_fft,p_fft); // uzywaj istniejacego
...
polar(m_fft,m_fft); // operacja w miejscu
}
```

Mat - pamięć i kopiowanie

Ponieważ Mat jest obiektem klasy to sam zarządza buforem

```
// create a big 8Mb matrix
Mat A(1000, 1000, CV_64F);

// create another header for the same matrix;
// this is instant operation, regardless of the matrix size.
Mat B = A;
// create another header for the 3-rd row of A; no data is copied either
Mat C = B.row(3);
// now create a separate copy of the matrix
Mat D = B.clone();
// copy the 5-th row of B to C, that is, copy the 5-th row of A
// to the 3-rd row of A.
B.row(5).copyTo(C);
// now let A and D share the data; after that the modified version
// of A is still referenced by B and C.
A = D;
// now make B an empty matrix (which references no memory buffers),
// but the modified version of A will still be referenced by C,
// despite that C is just a single row of the original A
B.release();

// finally, make a full copy of C. In result, the big modified
// matrix will be deallocated, since it's not referenced by anyone
C = C.clone();
```

Mat - dostęp do piksela

Iterator

```
vector<Mat> planes;
split (img_yuv, planes);
MatIterator_<uchar> it = planes[0].begin<uchar>(),
                    it_end = planes[0].end<uchar>();
for (; it != it_end; ++it)
{
    double v = *it*1.7 + rand();
    *it = saturate_cast<uchar>(v*v/255.);
}
```

Mat - dostęp do piksela

Dostęp do linii lub do elementu

```
vector<Mat> planes;
split (img_yuv, planes);
for( int y = 0; y < img_yuv.rows; y++ )
{
    uchar* Uptr = planes[1].ptr<uchar>(y);
    for( int x = 0; x < img_yuv.cols; x++ )
    {
        Uptr[x] = saturate_cast<uchar>((Uptr[x]-128)/2 + 128);
        uchar& Vxy = planes[2].at<uchar>(y, x);
        Vxy = saturate_cast<uchar>((Vxy-128)/2 + 128);
    }
}
```

Mat - dostęp do piksela

Dostęp elementu

```
Mat image = imread("../QtPobr/Lena.png");
for(int x = 100; x < 300; x++)
    for(int y=100; y <200; y++)
        image.at<Vec3b>(y,x)[0]=0;
```

jeszcze łatwiej

```
Mat_<Vec3b> image = imread("../QtPobr/Lena.png");
for(int x = 100; x < 300; x++)
    for(int y=100; y <200; y++)
        image(y,x)[0]=0;
```

Konwersja przestrzeni barw

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

Gdzie kody przestrzeni

```
#define CV_BGR2BGRA 0
#define CV_RGB2RGBA CV_BGR2BGRA
...
#define CV_BGR2GRAY 6
#define CV_RGB2GRAY 7
...
#define CV_BGR2YCrCb 36

#define CV_YCrCb2RGB 39

#define CV_BGR2HSV 40
#define CV_RGB2HSV 41

#define CV_BGR2Lab  44
#define CV_RGB2Lab  45
...
#define CV_BGR2HLS 52
#define CV_HLS2RGB 61
```

Skalowanie, przesunięcie, zmiana typu

```
void Mat::convertTo(Mat& m, int rtype, double alpha=1, double beta=0) co
```

$$m(x,y) = saturate\_cast < rType > (\alpha(*this)(x,y) + \beta) \qquad (1)$$

Gdzie rtype dane jest stałymi

```
#define CV_CN_MAX 64
#define CV_CN_SHIFT 3
#define CV_DEPTH_MAX (1 << CV_CN_SHIFT)

#define CV_8U 0
#define CV_8S 1
#define CV_16U 2
#define CV_16S 3
#define CV_32S 4
#define CV_32F 5
#define CV_64F 6
#define CV_USRTYPE1 7

#define CV_MAT_DEPTH_MASK (CV_DEPTH_MAX - 1)
#define CV_MAT_DEPTH(flags) ((flags) & CV_MAT_DEPTH_MASK)

#define CV_MAKETYPE(depth,cn) (CV_MAT_DEPTH(depth) + (((cn)-1) << CV_CN_SHIFT))
#define CV_MAKE_TYPE CV_MAKETYPE

#define CV_8UC1 CV_MAKETYPE(CV_8U,1)
```

Skalowanie, przesunięcie, zmiana typu

```
void Mat::convertTo(Mat& m, int rtype, double alpha=1, double beta=0) co
```

Gdzie rtype dane jest stałymi

```
#define CV_CN_MAX 64
#define CV_CN_SHIFT 3
#define CV_DEPTH_MAX (1 << CV_CN_SHIFT)

#define CV_8U 0
#define CV_8S 1
#define CV_16U 2
#define CV_16S 3
#define CV_32S 4
#define CV_32F 5
#define CV_64F 6
#define CV_USRTYPE1 7

#define CV_MAT_DEPTH_MASK (CV_DEPTH_MAX - 1)
#define CV_MAT_DEPTH(flags) ((flags) & CV_MAT_DEPTH_MASK)

#define CV_MAKETYPE(depth,cn) (CV_MAT_DEPTH(depth) + (((cn)-1) << CV_CN_SHIFT))
#define CV_MAKE_TYPE CV_MAKETYPE

#define CV_8UC1 CV_MAKETYPE(CV_8U,1)
#define CV_8UC3 CV_MAKETYPE(CV_8U,3)
#define CV_32FC1 CV_MAKETYPE(CV_32F,1)
```

dla obrazów typu float domyślny zakres to 0..1 (czyli jak nie przeskalujemy to funcja wyświetlająca zwraca białe okienko

Skalowanie, przesunięcie, zmiana typu

Operacja jest przycinana:

$$m(x, y) = saturate\_cast < rType > (\alpha(*this)(x, y) + \beta) \quad (2)$$

$$m(x, y) = min(max(value, 0), 255) \quad (3)$$

Analogiczna operacja dla innych typów całkowitych:

- CV_8U
- CV_8S
- CV_16U
- CV_16S
- CV_32S

Skalowanie, przesunięcie, zmiana typu

I zamiast wywołania funkcji można wykorzystać przeciążone operatory:

```
Mat rgb = ...
Mat gray = Mat(rgb.size(), CV_8UC1);
cvtColor(rgb,gray,CV_BGR2GRAY);
Mat fgray=gray.convertTo(gray, CV_32FC1, 1./255,0);
imshow("Lena1",fgray);
```

odpowiada:

```
Mat rgb = ...
Mat gray = Mat(rgb.size(), CV_8UC1);
cvtColor(rgb,gray,CV_BGR2GRAY);
Mat fgray=gray.convertTo(gray, CV_32FC1);
fgray = fgray / 255;
imshow("Lena1",fgray);
```

Filtrowanie

Ogólnie:

```
void filter2D (const Mat& src, Mat& dst, int ddepth, const Mat& kernel, Poi
```

Ale maskę trzeba odwrócić:

```
void flip (const Mat& src, Mat& dst, int flipCode)
```

i przesunąć punkt centralny do

$$(kernel.cols - anchor.x - 1, kernel.rows - anchor.y - 1) \quad (4)$$

mamy przygotowane funkcje filtrujące:

```
void GaussianBlur(const Mat& src, Mat& dst, Size ksize, double sigmaX, dou
void blur(const Mat& src, Mat& dst, Size ksize, Point anchor=Point(-1, -1
void Sobel(const Mat& src, Mat& dst, int ddepth, int xorder, int yorder, in
```

koniec