# 知识回顾Sobel算子

Sobel算子学习网站：

<http://www.opencv.org.cn/opencvdoc/2.3.2/html/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html#sobel-derivatives>

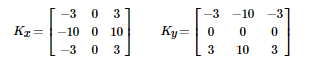
## Scharr 滤波器

Scharr滤波器，主要是配合Sobel算子而存在的。该函数与Sobel的区别在于，Scharr仅作用于大小为3的内核。

当ksize=-1时，卷积核大小3x3 时，使用 sobel 卷积核来计算并不是很精确，Sobel内核可能产生比较明显的误差【Sobel算子只是求取了导师的近似值而已】，此时常用 Scharr 卷积核来代替，使用3\*3 的Scharr滤波器，效果要比3\*3的Sobel滤波器好，具有和sobel算子一样的速度，但结果更为精确。

所以在使用3\*3滤波器时应该尽量使用Scharr滤波器（一般就用Sobel算子即可）。

如下：



而 Sharr 函数，本质上就是令 ksize = 3 且使用 Scharr 卷积核的 Sobel 函数。

Scharr ( src, dst, ddepth, dx, dy, scale , delta , borderType )

等价

Sobel ( src, dst, ddepth, dx, dy, CV\_SCHARR, scale , delta , borderType )

### Scharr函数

计算图像差分

void Scharr (

InputArray src, //Mat类型图像即可。

OutputArray dst,

int ddepth,

int dx,

int dy,

double scale = 1,

double delta = 0,

int borderType = BORDER\_DEFAULT

)

对于 Scharr 函数，要求 dx 和 dy 都 >= 0 且 dx + dy == 1，假如 dx 和 dy 都设为 1，则会抛出异常。

对于 Sobel 和 Scharr 函数，通常各自求其 x 和 y 方向的导数，然后通过加权来进行边缘检测。

### Scharr代码编程





#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

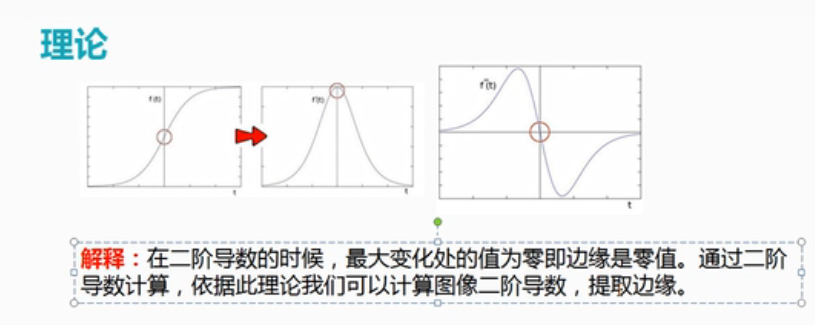
int main()

{



}

## Laplacian算子





对图像求二阶导 ，Laplace其实利用Sobel算子的运算，它通过Sobel算子运算出图像在x方向和y方向的导数，来得到我们载入图像的拉普拉斯变换结果。

Laplace的说明，

The function calculates the Laplacian of the source image by adding up the second x and y

. derivatives calculated using the Sobel operator:

拉普拉斯算子的定义如下：

其中，Sobel算子（ksize>1），如下：



（ksize==1）

Laplacian函数采用3\*3的孔径；

0 1 0

1 -4 1

0 1 0

### 编程实现Laplacian的思想

#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

Mat MyLaplacian (Mat &srcImage);

int main()

{

Mat src = imread("e:\\house.jpg",1);

//Mat src = imread("e:\\shuiguo.png",1);

imshow("src",src);

GaussianBlur(src,src,Size(3,3),0,0,BORDER\_DEFAULT);

Mat srcgray;

cvtColor(src,srcgray,COLOR\_BGR2GRAY);

Mat dst= MyLaplacian (srcgray);

imshow("dst", dst);

waitKey(0);

return 0;

}

Mat MyLaplacian (Mat &srcImage)

{

Mat dstImage = srcImage.clone(); // Mat dstImage(srcImage.size(),srcImage.type());

int thresh =10;

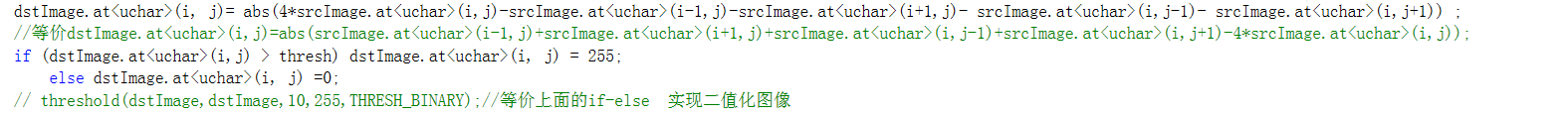
for (int i = 1; i < dstImage.rows - 1; i++)//边界提取

{

for (int j = 1; j < dstImage.cols -1; j++)

{

//补充代码



}

}

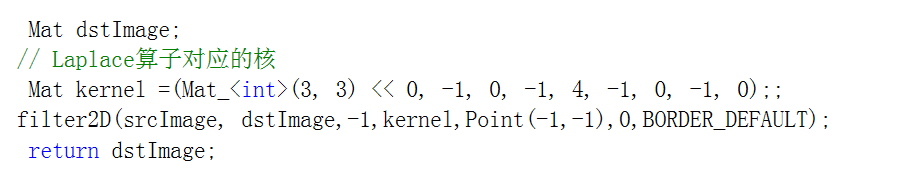
return dstImage;

}

### 编程采用filter2D实现Laplacian的思想

Mat MyLaplacian (Mat &srcImage)

{



}

### Laplacian函数：

计算图像经过拉普拉斯变换后的结果

void Laplacian( InputArray src, OutputArray dst, int ddepth,

int ksize = 1, double scale = 1, double delta = 0,

int borderType = BORDER\_DEFAULT );

src：需要为Mat类型的单通道8位图像

Ddepth：目标图像的深度

ksize ：用于计算二阶导数的滤波器的孔径大小，大小必须是正奇数， 有默认值1

scale： 计算拉普拉斯可选比例因子，有默认值1

delta：加到输出像素的值，默认为0

borderType：边界模式。默认值BORDER\_DEFAULT

### Laplacian代码编程

#include<opencv2\opencv.hpp>

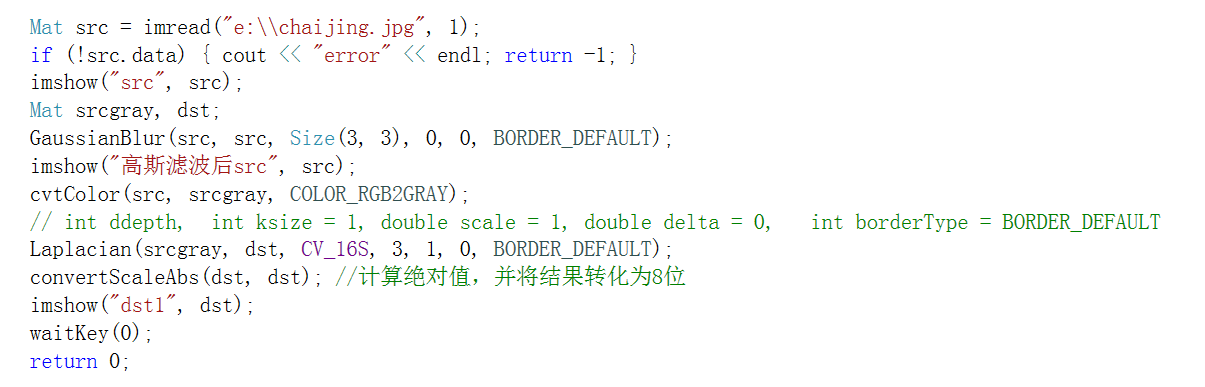
#include<iostream>

using namespace std;

using namespace cv;

int main()

{



}

### 学习Laplacian网站：<http://www.opencv.org.cn/opencvdoc/2.3.2/html/doc/tutorials/imgproc/imgtrans/laplace_operator/laplace_operator.html>

# Canny算法

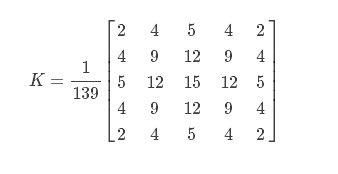
John F.Canny于1986年开发出来的一个多级边缘检测算法。Canny边缘检测算法被很多人推崇为当今最优秀的边缘检测算法 ， opencv中提供了Canny函数

### Canny边缘检测的步骤

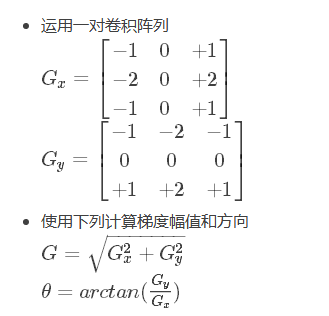
使用高斯滤波器对图像进行去噪、计算梯度、在边缘上使用非最大抑制(NMS)、在检测到的边缘上使用双阈值去除假阳性(false positive)，最后还会分析所有的边缘及其之间的连接，以保留真正的边缘并消除不明显的边缘。

（1）消除噪声

高斯平滑滤波卷积降噪（size = 5）



（2）计算梯度幅值和方向



（3）非极大值抑制

排除非边缘像素，仅仅保留一些细线条

（4）滞后阈值

若某一像素位置的幅值超过高阈值，该像素被保留为边缘像素

若某一像素位置的幅值小于低阈值，该像素被排除

若某一像素位置的幅值在两个阈值之间，该像素仅仅在连接一个高于高阈值的像素时被保留

### Canny函数参数功能

void Canny (

InputArray image, // 输入图像 (8位)

OutputArray edges, // 输出图像 (单通道，8位)

double threshold1, // 下阈值，低阈值

double threshold2, // 上阈值，高阈值

int apertureSize = 3,//Sobel算子内核(kSize)大小

bool L2gradient = false，

)

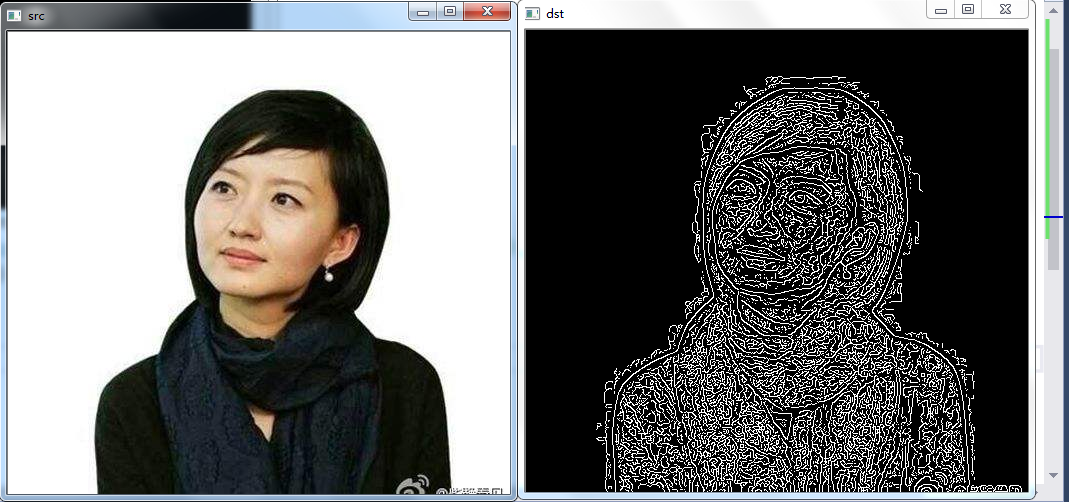
threshold1,threshold2 中较小的阈值用于边缘连接，较大的阈值用来控制强边缘的初始段，推荐的高低阈值比在2:1~到3:1之间。这二个阈值中当中的小阈值用来控制边缘连接，大的阈值用来控制强边缘的初始分割即如果一个像素的梯度大与上限值，则被认为是边缘像素，如果小于下限阈值，则被抛弃。如果该点的梯度在两者之间则当这个点与高于上限值的像素点连接时我们才保留，否则删除。

apertureSize第五个参数表示Sobel 算子大小，默认为3即表示一个3\*3的矩阵。Sobel 算子与高斯拉普拉斯算子都是常用的边缘算子

L2gradient默认 flase，表示用L1范数(表示图像梯度强度的计算采用近似形式:直接将两个方向导数的绝对值相加）;为真表示使用更精确的L2范数进行计算(表示采用更精确的形式:两个方向的倒数的平方再开放）

### Canny代码编程

canny算法提取的轮廓图，把细致的额轮廓都提取出来 ！



#include<opencv2\opencv.hpp>

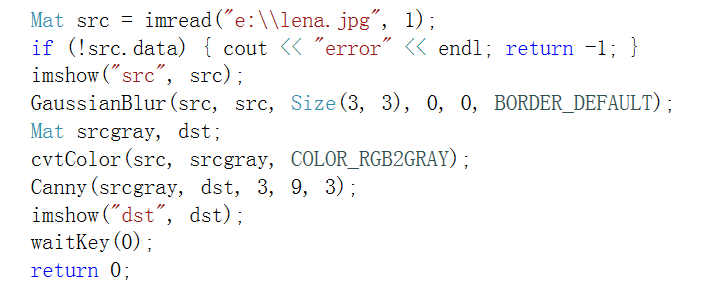
#include<iostream>

using namespace std;

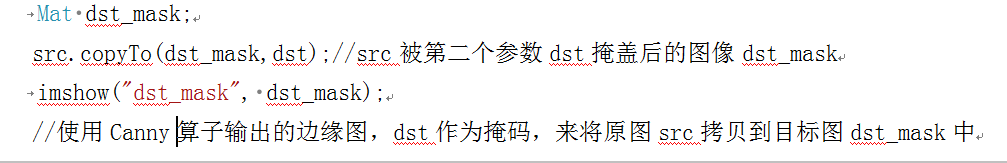
using namespace cv;

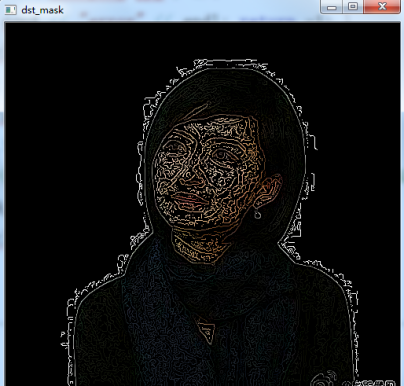
int main()

{

}

代码扩展：





### 学习Canny网站

http://www.opencv.org.cn/opencvdoc/2.3.2/html/doc/tutorials/imgproc/imgtrans/canny\_detector/canny\_detector.html

## 编程：滑块（低阈值：0~120，【推荐高低阈值比在2:1~到3:1之间】 ）+Canny代码

