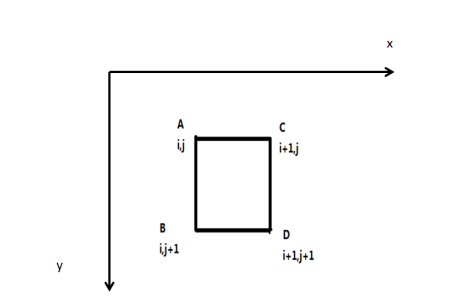
# 图像的差分运算

差分就是相减的意思，你可以对两幅图像进行对应位置的像素值相减操作，也可以在一幅图像里对某个区域内的像素值进行相减操作，得到所说的差分图像。

二维离散图像f(i,j)在x方向的前向差分f(i+1,j)-f(i,j)，y方向的前向差分f(i,j+1)-f(i,j)

二维离散图像f(i,j)在x方向的逆向差分f(i,j)-f(i-1,j)，y方向的逆向差分f(i,j)-f(i,j-1)

前向差分通常是微分在离散的函数中的等效运算。



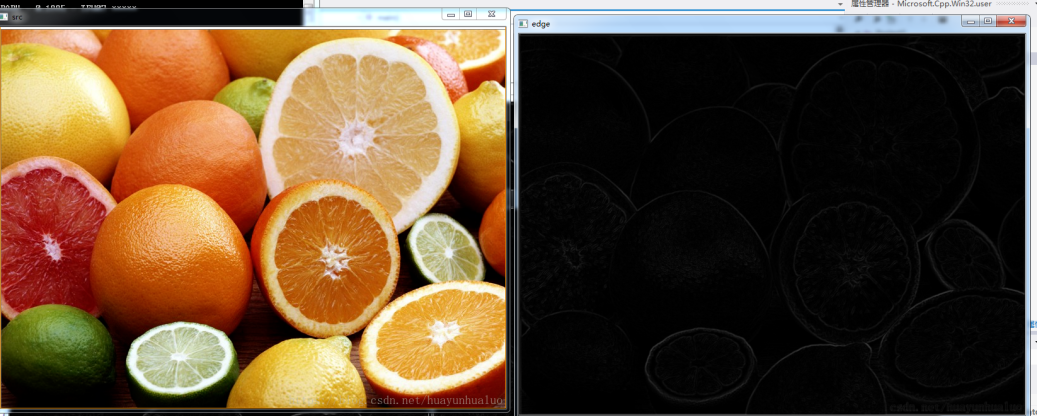
差分运算通过求灰度图像变换剧烈处的一阶微分算子的极值来检测奇异点，通过奇异点的值进一步设定阈值就可以获得边缘二值化图像。差分检测边缘中差分水平或垂直都与边缘方向正交，因此在实际的应用场景中，常常将边缘检测分为水平边缘，垂直边缘和对角线边缘，差分运算边缘检测定义方向模板如下；

水平边缘

垂直边缘

对角线边缘

## 差分边缘检测实现代码：



#include <iostream>

#include <opencv2/opencv.hpp>

using namespace std;

using namespace cv;

void diffOperation(Mat srcImage,Mat&edgeXImage,Mat &edgeYImage);

int main()

{

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

imshow("src", src);

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

cvtColor(src,src,COLOR\_BGR2GRAY);

Mat edgeX(src.size(), src.type());

Mat edgeY(src.size(), src.type());

Mat edge(src.size(), src.type());

diffOperation(src, edgeX, edgeY);

imshow("edgeX", edgeX);

imshow("edgeY", edgeY);

addWeighted(edgeX, 0.5, edgeY, 0.5, 0, edge);

imshow("edge", edge);

waitKey(0);

return 0;

}

void diffOperation(Mat srcImage, Mat&edgeXImage, Mat &edgeYImage)

{

Mat temp = srcImage.clone();

for (int i = 0; i < temp.rows-1; i++)

{

for (int j = 0; j < temp.cols-1; j++)

{

//计算垂直边缘

edgeXImage.at<uchar>(i, j) = abs(temp.at<uchar>(i + 1, j) - temp.at<uchar>(i, j));

//计算水平边缘

edgeYImage.at<uchar>(i, j) = abs(temp.at<uchar>(i , j+ 1) - temp.at<uchar>(i, j));

}

}

}

# Matlab实现常用的边缘检测算子：

clc;

clear;

f=imread("cell.tif");

figure,imshow(f);

[g\_roberts]= edge(f, ‘roberts ');

figure,imshow(g\_roberts);

[g\_prewitt]=edge(f,'prewitt');

figure,imshow(g\_presitt);

[g\_sobel]=edge(f,'sobel');

figure,imshow(g\_sobel);

[g\_log]=edge(f,'log');

figure,imshow(g\_log);

[g\_canny]=edge(f,'edge');

figure,imshow(g\_canny);

# 边缘检测

边缘检测是图像处理与计算机视觉中的重要技术之一，其目的是检测识别出图像中亮度变化剧烈的像素点构成的集合。

边缘检测大大减少了源图像的数据量，剔除了与目标不相干的信息，保留了图像重要的结构属性。

图像边缘的正确检测有利于分析目标检测、定位及识别。

通常目标物体形成边缘存在以下几种情形：

<1>目标物呈现在图像的不同物体平面上，深度不连续

<2>目标物本身平面不同，表面方向不连续

<3>目标物材料不均匀，表面反射光不同

<4>目标物受外部场景光影响不一

根据边缘形成的原因，对图像的各像素点进行求一阶微分或二阶微分可以检测出灰度变化明显的点。边缘检测算子是利用图像边缘的突变性质来检测边缘的，通常情况下将边缘检测分为以下三个类型：

<1>一阶微分为基础的边缘检测，通过计算图像的梯度值来检测图像边缘，如Sobel算子、Prewitt算子及差分边缘检测。

<2>二阶微分为基础的边缘检测，通过寻求二阶导数中的过零点来检测边缘，如拉普拉斯算子、高斯拉普拉斯算子、Canny算子边缘检测。

<3>混合一阶与二阶微分为基础的边缘检测，综合利用一阶微分与二阶微分特征，如Marr-Hidreth算子。

## 边缘检测的一般步骤：

1、滤波—消除噪声: 边缘检测算法主要是基于图像强度的一阶和二阶导数，但是导数对于噪声很敏感，因此需要采用滤波器来改善与噪声有关的边缘检测器的性能

2、增强—使边界轮廓更加明显: 增强边缘的基础是确定图像各点邻域强度的变化值。增强算法可以将灰度点邻域强度值有显著变化的点凸显出来

3、检测—选出边缘点:邻域中有很多的点的梯度值较大，但是在特定的应用中，这些点并不是要找的边缘点，需要取舍

## 边缘检测评价标准

* 低错误率：标识出尽可能多的实际边缘，同时尽可能地减少噪声产生的误报。
* 高定位性：标识出的边缘要与图像中的实际边缘尽可能接近
* 最小响应：图像中的边缘只能标识一次，并且可能存在的图像噪声不应标识为边缘

## Roberts边缘算子理论知识

1963年，Roberts（罗伯茨）提出了这种寻找边缘的算子。

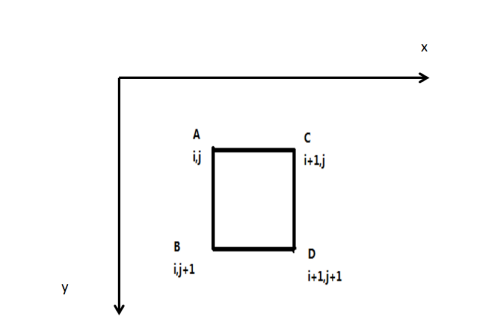
罗伯茨算子、Roberts算子是一种最简单的算子，是一种利用局部差分算子寻找边缘的算子，他采用对角线方向相邻两象素之差近似梯度幅值检测边缘。

检测垂直边缘的效果好于斜向边缘，定位精度高，对噪声敏感,无法抑制噪声的影响。

Roberts边缘算子是一个2\*2的模板，采用的是对角方向相邻的两个像素之差。

从图像处理的实际效果来看，边缘定位较准，对噪声敏感。适用于边缘明显且噪声较少的图像分割。

Roberts算子图像处理后结果边缘不是很平滑。经分析，由于Robert算子通常会在图像边缘附近的区域内产生较宽的响应，故采用上述算子检测的边缘图像常需做细化处理，边缘定位的精度不是很高。



i: 0~最大行数-1 j: 0~最大列数-1

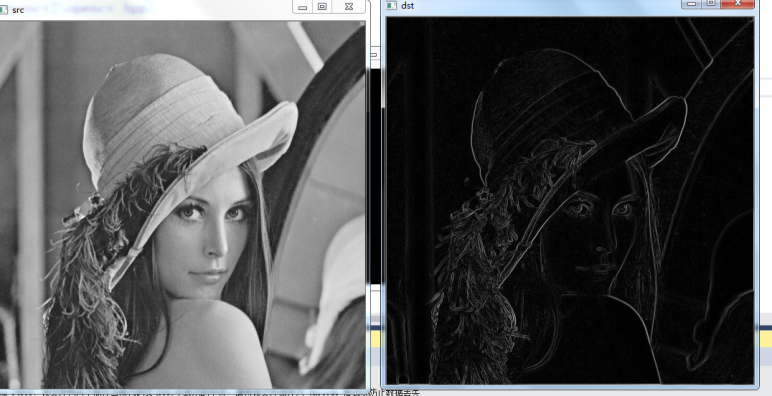
对原始图像f(i,j),Roberts边缘检测输出图像 g(i,j)，图像的Roberts边缘检测可以用下面的公式：

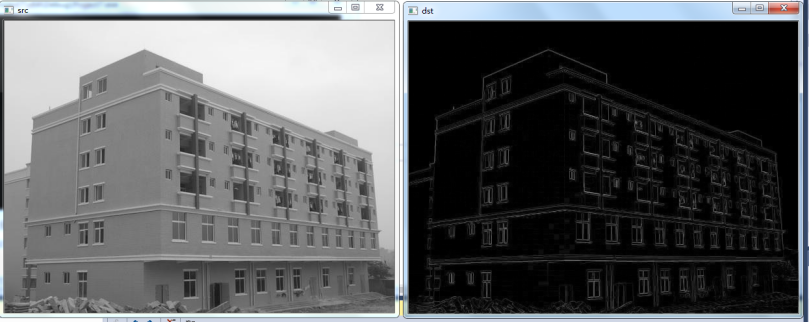


可得Roberts算子模板为  

或近似计算







### Roberts编程

#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

Mat roberts(Mat &srcImage);

int main()

{

Mat src = imread("e:\\lena.jpg",0);

imshow("src",src);

Mat dst=roberts(src);

imshow("dst", dst);

waitKey(0);

return 0;

}

Mat roberts(Mat &srcImage)

{

return dstImage;

}

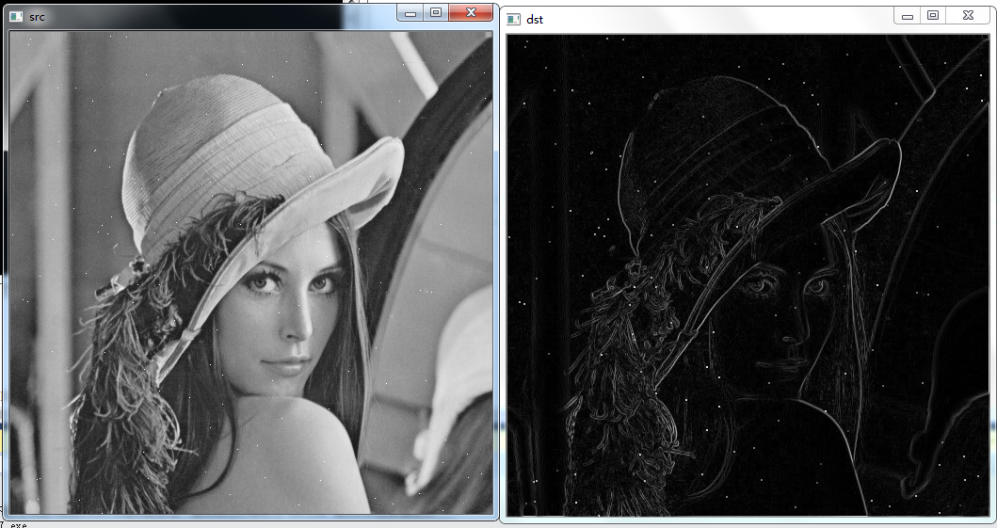
把Mat src = imread("e:\\lena.jpg",0);

改为下面几句话尝试编程效果：

//Mat src = imread("e:\\house.jpg",0);

//Mat src = imread("e:\\lena\_salt.jpg",0);//Roberts算子对噪声的抑制能力较弱，对噪声敏感

//Mat src = imread("e:\\road.png",0);



Roberts算子检测垂直边缘的效果好于斜向边缘，定位精度高，对噪声敏感,无法抑制噪声的影响。Roberts算法适用于边缘明显而且噪声较少的图像分割，在应用中经常使用算法来提取道路

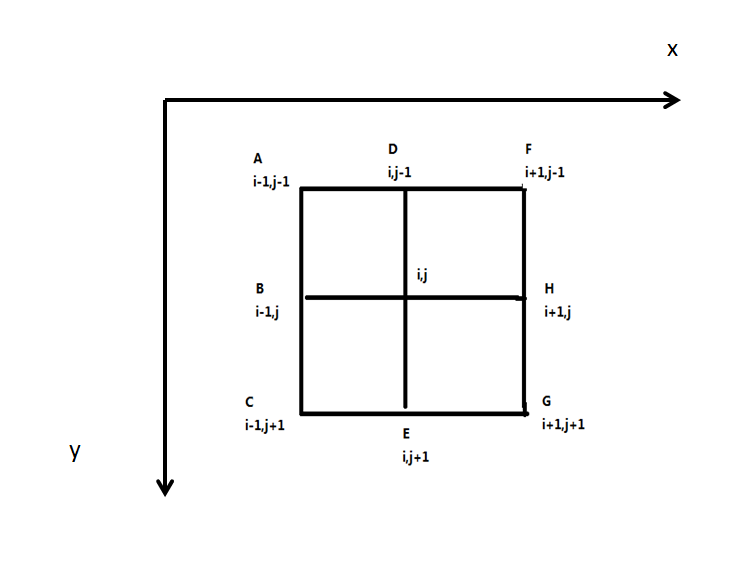
Roberts边缘算子是一个2x2的模板，采用的是对角方向相邻的两个像素之差。从图像处理的实际效果来看，边缘定位较准，对噪声敏感。适用于边缘明显且噪声较少的图像分割。

## 基本边缘检测算子Prewitt

Prewitt 算子利用像素点上下、左右邻点灰度值，在边缘处达到极值检测边缘，对噪声具有平滑作用。Prewitt边缘算子不仅能检测边缘点，而且能够抑制噪声的影响，因此对灰度和噪声较多的图像处理的较好。

由于边缘像素的灰度值与其邻域点像素的灰度值有显著不同，在实际应用中，通常采用微分算子和模板匹配方法检测图像的边缘，Prewitt边缘算子的卷积如下所示，图像中每个像素都与这俩核做卷积，取最大值作为输出产生一副边缘图像。

对原始图像f(x,y)，Prewitt边缘检测输出图像是G，图像的Prewitt边缘检测可表示为：





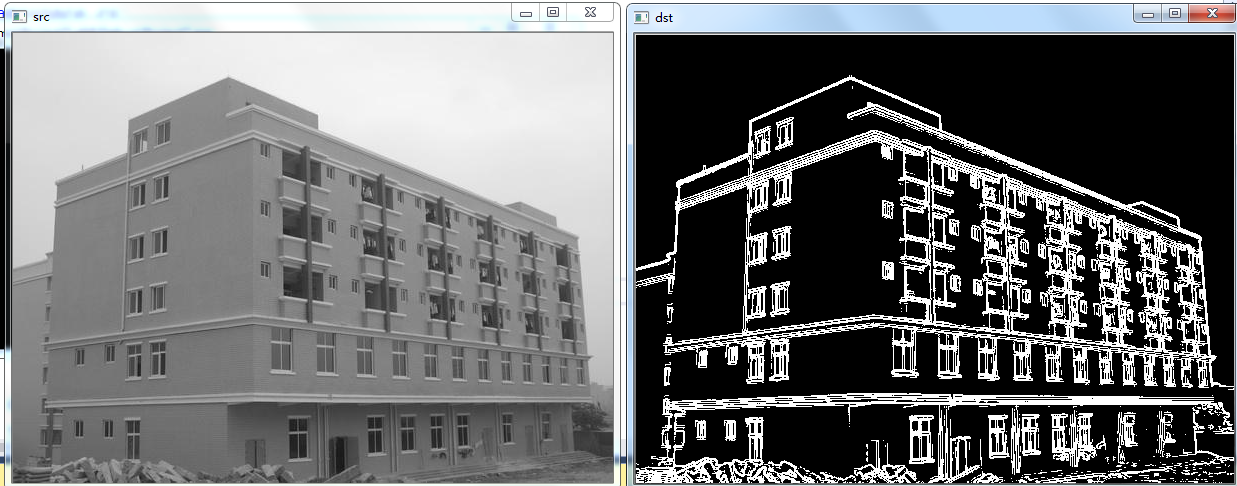


对于最后输出的边缘图像，可根据G=max(Gx，Gy) 或G=Gx+Gy得到，凡灰度新值大于或等于阈值的像素点就认为是边缘点，即选择合适的阈值T，若G >=T，则认为是边缘点。根据上面的公式，可得的模板为：

水平方向的算子：

垂直方向的算子：

### Prewitt代码



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

Mat prewitt(Mat &srcImage);

int main()

{

Mat src = imread("e:\\house.jpg",0);

imshow("src",src);

Mat dst= prewitt(src);

imshow("dst", dst);

waitKey(0);

return 0;

}

Mat prewitt(Mat &srcImage)

{

Mat dstImage = srcImage.clone();

int thresh =60;

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

{

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

{

int t1,t2,t;

t1= abs(srcImage.at<uchar>(i-1,j-1)+ srcImage.at<uchar>(i-1,j)+ srcImage.at<uchar>(i-1,j+1)- srcImage.at<uchar>(i+1,j-1)- srcImage.at<uchar>(i+1,j)- srcImage.at<uchar>(i+1,j+1)) ;

t2= abs(srcImage.at<uchar>( i-1,j+1) + srcImage.at<uchar>(i , j+1) + srcImage.at<uchar>(i + 1, j + 1) - srcImage.at<uchar>(i - 1, j - 1) - srcImage.at<uchar>(i , j-1) - srcImage.at<uchar>(i + 1, j -1));

t = t1 + t2;

if (t > thresh) dstImage.at<uchar>(i, j) = 255;

else dstImage.at<uchar>(i, j) =0;

}

}

return dstImage;

}

## 基本边缘检测算子 Sobel【索贝尔算子】

Sobel算子是一个主要用于边缘检测的离散微分算子(discrete differentiation operator)，它结合高斯平滑和微分求导。用来计算图像灰度函数的近似函数.

Sobel算子是一个主要用于边缘检测的离散微分算子(discrete differentiation operator), Sobel算子结合了高斯平滑和微分求一阶，用来计算图像灰度函数的近似梯度，是一种常用的边缘检测算子，对噪声具有平滑作用，提供较为精确的边缘方向信息，边缘定位精度不够高。当对精度要求不是很高时，是一种较为常用的边缘检测方法。

Sobel 算子有两个，一个是检测水平边缘的 ；另一个是检测垂直边缘的 。

Sobel算子对于象素的位置的影响做了加权，可以降低边缘模糊程度，因此效果更好。

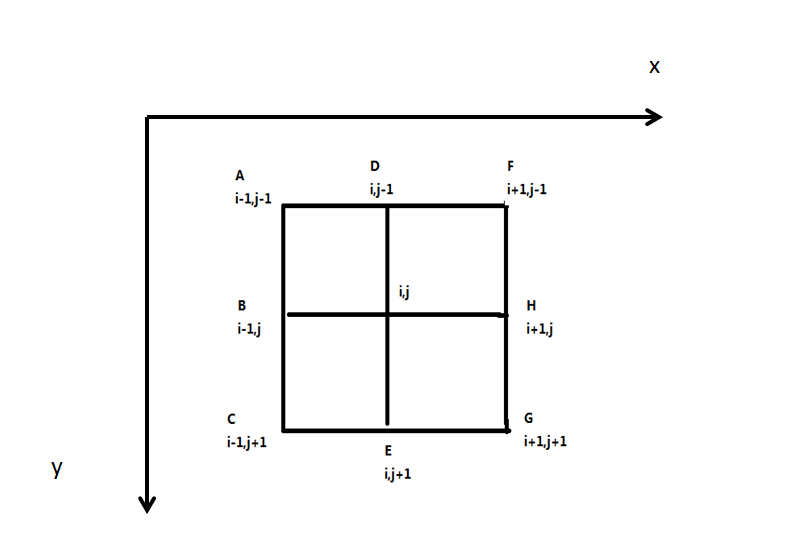
Sobel算子根据像素点上下、左右邻点灰度加权差，在边缘处达到极值这一现象检测边缘。

假定输入图像矩阵为f，卷积核大小为 3x3

Gx，Gy代表利用模板对原图像卷积的结果；

水平一阶导数 Gx , 表示经纵向边缘检测的图像

垂直一阶导数 Gy , 表示经横向边缘检测的图像



水平变化：将图像f与一个奇数大小的内核Gx 进行卷积，比如，当内核大小为3时，Gx的计算结果：

Gx =-1\*f(i-1, j-1) +0\*f(i,j-1) + 1\*f(i+1,j-1) +(-2)\*f(i-1,j) + 0\*f(i,j)+2\*f(i+1,j)+(-1)\*f(i-1,j+1) + 0\*f(i,j+1) + 1\*f(i+1,j+1)

垂直变化：将图像f与一个奇数大小的内核Gy进行卷积，比如，当内核大小为3时，Gy的计算结果：

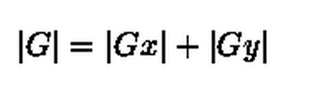
Gy =(-1)\*f(i-1,j+1) + (-2)\*f(i,j+1) + (-1)\*f(i+1, j+1) +0\*f(i-1,j) 0\*f(i,j) + 0\*f(i+1,j) +1\*f(i-1, j-1)+ 2\*f(i,j-1)+ 1\*f(i+1,j-1)

该算子包含两组3x3的矩阵，分别为横向及纵向，将之与图像作平面卷积，即可分别得出横向及纵向的亮度差分近似值。

在图像上的每一个点，结合下面公式，求出近似梯度， 输出的图像矩阵 **G** 为：

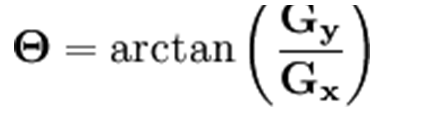


通常，为了提高效率，使用更简单的公式代替， 使用不开平方的近似值：



对于原图像中的每一个像素，在3\*3的模板中进行上述的卷积，得到GX、GY，则最后该像素的灰度值近似为： G = |GX|+|GY|

如果梯度G大于某一阀值,则认为该点(x,y)为边缘点。

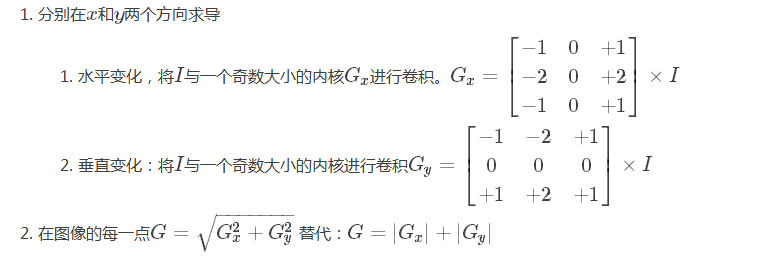


如果以上的角度Θ等于零，即代表图像该处拥有纵向边缘，左方较右方暗。

上述的处理可以同时进行两个方向的处理，当需要突出图像某一个方向的边缘信息时，也可以只进行其中一个方向的处理。

### 编写代码实现Sobel算子的功能

Sobel算子的计算过程：



使用扩展的算子，来计算一阶、二阶、三阶或者混合图像差分。

#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

Mat sobel(Mat &srcImage);

int main()

{

Mat src = imread("e:\\house.jpg",0);

imshow("src",src);

Mat dst= sobel(src);

imshow("dst", dst);

waitKey(0);

return 0;

}

Mat sobel(Mat &srcImage)

{

Mat dstImage = srcImage.clone();

int thresh =60;

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

{

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

{

int t1,t2,t;

t1= abs(srcImage.at<uchar>(i-1,j+1)+ 2\*srcImage.at<uchar>(i ,j+1)+ srcImage.at<uchar>(i+1,j+1)- srcImage.at<uchar>(i-1,j-1)- 2\*srcImage.at<uchar>(i,j-1)- srcImage.at<uchar>(i+1,j-1)) ;

t2= abs(srcImage.at<uchar>( i-1,j-1) + 2\*srcImage.at<uchar>(i-1, j) + srcImage.at<uchar>(i -1, j + 1) - srcImage.at<uchar>(i+ 1, j - 1) - 2\*srcImage.at<uchar>(i+1 , j) - srcImage.at<uchar>(i + 1, j +1));

t = t1 + t2;

if (t > thresh) dstImage.at<uchar>(i, j) = 255;

else dstImage.at<uchar>(i, j) =0;

}

}

return dstImage;

}

### Sobel函数

使用扩展的算子，来计算一阶、二阶、三阶或者混合图像差分。

void Sobel (

InputArray src, // 输入图像，Mat类型即可

OutputArray dst, // 输出图像，大小与原图像相同

int ddepth, // 输出图像深度，-1 表示等于 src.depth()

int dx , // 水平方向的阶数，1或0

int dy , // 垂直方向的阶数，1或0

int ksize = 3, // 卷积核的大小，常取 1, 3, 5, 7 等奇数

double scale = 1, // 缩放因子，应用于计算结果，scale = 1，表示计算结果不缩放；

double delta = 0, // 增量数值，应用于计算结果，delta = 0，表示计算结果无增量。  
 int borderType = BORDER\_DEFAULT // 边界处理模式  
)

int ddepth：输出图像的深度，针对不同的输入图像，输出目标图像有不同的深度，具体组合如下：

- 若src.depth() = CV\_8U, 取ddepth =-1/CV\_16S/CV\_32F/CV\_64F

- 若src.depth() = CV\_16U/CV\_16S, 取ddepth =-1/CV\_32F/CV\_64F

- 若src.depth() = CV\_32F, 取ddepth =-1/CV\_32F/CV\_64F

- 若src.depth() = CV\_64F, 取ddepth = -1/CV\_64F

注：ddepth =-1时，代表输出图像与输入图像相同的深度。

int dx：int类型dx，x 方向上的差分阶数，但不超过 2

int dy：int类型dy，y 方向上的差分阶数 但不超过 2

其中，dx=1，dy=0，表示计算X方向的导数，检测出的是垂直方向上的边缘；Gx

dx=0，dy=1，表示计算Y方向的导数，检测出的是水平方向上的边缘。 Gy

int ksize：为进行边缘检测时的模板大小为ksize\*ksize，取值为1、3、5和7，其中默认值为3。特殊情况：ksize=1时，采用的模板为3\*1或1\*3。

delat = 0，表示计算结果无增量。

### Sobel编程

Sobel核心代码：

步骤：



Sobel 示例 ，使用 addWeighted 函数，来加权合成 x 和 y 方向上各自的一阶导数【是一种近似的估计】





#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{



}

## 辅助学习：convertScaleAbs()函数

使用线性变换转换输入数组元素成8位无符号整数

void convertScaleAbs(InputArray src, OutputArray dst,

double alpha = 1, double beta = 0);

dst=saturate<uchar>(src\*alpha +beta)



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

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

if (!src.data) { cout << "error" << endl; return -1; }

imshow("src", src);

Mat dst;

convertScaleAbs(src, dst, 1.5, 20);

imshow("dst", dst);

waitKey(0);

return 0;

}

## abs是绝对值的意思，abs(A)应该是求A中每个元素的绝对值