1.加载、修改、保存图像

- 加载图像（用cv::imread）
- 修改图像 (cv::cvtColor)
- 保存图像(cv::imwrite)
- 代码演示

加载图像（用cv::imread）

- imread功能是加载图像文件成为一个Mat对象，其中第一个参数表示图像文件名称
- 第二个参数，表示加载的图像是什么类型，支持常见的三个参数值
- IMREAD_UNCHANGED (<0) 表示加载原图，不做任何改变
- IMREAD_GRAYSCALE (0)表示把原图作为灰度图像加载进来
- IMREAD_COLOR (>0) 表示把原图作为RGB图像加载进来

注意： OpenCV支持JPG、 PNG、 TIFF等常见格式图像文件加载

显示图像 (cv::namedWindos 与cv::imshow)

- namedWindos功能是创建一个OpenCV窗口，它是由OpenCV自动创建与释放，你无需取销毁它。
- 常见用法namedWindow("Window Title", WINDOW_AUTOSIZE)
- WINDOW_AUTOSIZE会自动根据图像大小，显示窗口大小，不能人为改变窗口大小
- WINDOW_NORMAL,跟QT集成的时候会使用，允许修改窗口大小。
- imshow根据窗口名称显示图像到指定的窗口上去，第一个参数是窗口名称，第二参数是Mat对象

修改图像 (cv::cvtColor)

- cvtColor的功能是把图像从一个彩色空间转换到另外一个色彩空间，有三个参数，第一个参数表示源图像、第二参数表示色彩空间转换之后的图像、第三个参数表示源和目标色彩空间如：COLOR_BGR2HLS 、COLOR_BGR2GRAY 等
- cvtColor(image, gray_image, COLOR_BGR2GRAY);

保存图像(cv::imwrite)

- 保存图像文件到指定目录路径
- 只有8位、16位的PNG、JPG、Tiff文件格式而且是单通道或者三通道的BGR的图像才可以通过这种方式保存
- 保存PNG格式的时候可以保存透明通道的图片
- 可以指定压缩参数

图像加载、修改、保存的代码演示

- 代码演示程序

2.矩阵的掩膜操作

- 获取图像像素指针
- 掩膜操作解释
- 代码演示

获取图像像素指针

- CV_Assert(myImage.depth() == CV_8U);
- Mat.ptr<uchar>(int i=0) 获取像素矩阵的指针，索引i表示第几行，从0开始计行数。
- 获得当前行指针const uchar* current= myImage.ptr<uchar>(row);
- 获取当前像素点P(row, col)的像素值 p(row, col) =current[col]

像素范围处理saturate_cast<uchar>

- saturate_cast<uchar>（-100），返回 0。
- saturate_cast<uchar>（288），返回255
- saturate_cast<uchar>（100），返回100
- 这个函数的功能是确保RGB值得范围在0~255之间

掩膜操作实现图像对比度调整

-红色是中心像素，从上到下，从左到右对每个像素做同样的处理操作，得到最终结果就是对比度提高之后的输出图像Mat对象

矩阵的掩膜操作十分简单，根据掩膜来重新计算每个像素的像素值，掩膜(mask 也被称为Kernel)

通过掩膜操作实现图像对比度提高。

$$I(i,j) = 5 * I(i,j) - [I(i-1,j) + I(i+1,j) + I(i,j-1) + I(i,j+1)]$$

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

代码实现

```
int main(int argc, char** argv)
{
    Mat myImage = imread("D:/test.jpg");
    CV_Assert(myImage.depth() == CV_8U);
    namedWindow("mask_demo", CV_WINDOW_AUTOSIZE);
    imshow("mask_demo", myImage);

    // clone current image
    Mat resultImage;
    myImage.copyTo(resultImage);
    int nchannels = myImage.channels();
    int height = myImage.rows;
    int cols = myImage.cols;
    int width = myImage.cols * nchannels;
    for (int row = 1; row < height - 1; row++) {
        const uchar* previous = myImage.ptr<uchar>(row - 1);
        const uchar* current = myImage.ptr<uchar>(row);
        const uchar* next = myImage.ptr<uchar>(row + 1);
        uchar* output = resultImage.ptr<uchar>(row);
        for (int col = nchannels; col < nchannels * (myImage.cols - 1); col++) {
            *output = saturate_cast<uchar>(5 * current[col] - previous[col] - next[col] - current[col - nchannels] - current[col + nchannels]);
            output++;
        }
    }

    namedWindow("mask_result", CV_WINDOW_AUTOSIZE);
    imshow("mask_result", resultImage);

    // 关闭
    waitKey(0);
    return 0;
}
```

函数调用filter2D功能

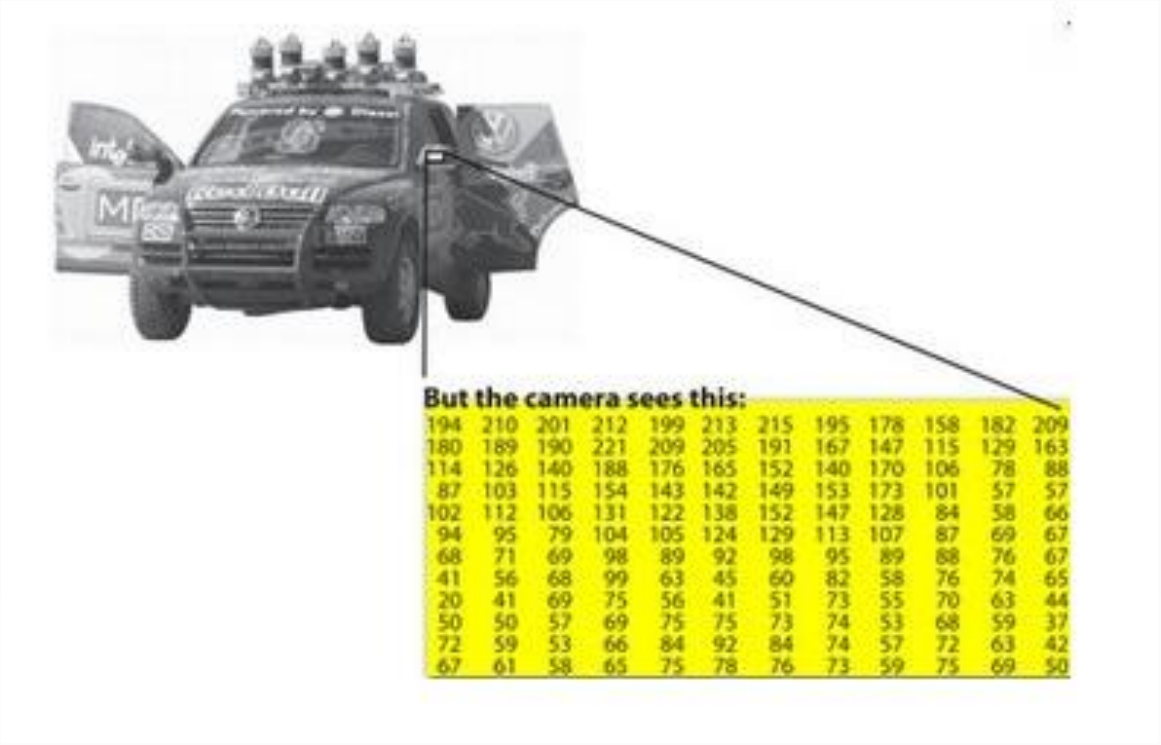
1. 定义掩膜： Mat kernel = (Mat_<char>(3,3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);
2. filter2D(src, dst, src.depth(), kernel);其中src与dst是Mat类型变量、src.depth表示位图深度，有32、24、8等。

```
Mat kernel = (Mat_<char>(3,3) <<  0, -1,  0,
                                 -1,  5, -1,
                                  0, -1,  0);
t = (double)getTickCount();
filter2D( src, dst1, src.depth(), kernel );
t = ((double)getTickCount() - t)/getTickFrequency();
cout << "Built-in filter2D time passed in seconds:      " << t << endl;

imshow( "Output", dst1 );
```

3.Mat对象

- Mat对象与IplImage对象
- Mat对象使用
- Mat定义数组

Mat对象


But the camera sees this:
194 210 201 212 199 213 215 195 178 158 182 209
180 189 190 221 209 205 191 167 147 115 129 163
114 126 140 188 176 165 152 140 170 106 78 88
87 103 115 154 143 142 149 153 173 101 57 57
102 112 106 131 122 138 152 147 128 84 58 66
94 95 79 104 105 124 129 113 107 87 69 67
68 71 69 98 89 92 98 95 89 88 76 67
41 56 68 99 63 45 60 82 58 76 74 65
20 41 69 75 56 41 51 73 55 70 63 44
50 50 57 69 75 75 73 74 53 68 59 37
72 59 53 66 84 92 84 74 57 72 63 42
67 61 58 65 75 78 76 73 59 75 69 50

Mat对象与IplImage对象

- **Mat**对象OpenCV2.0之后引进的图像数据结构、自动分配内存、不存在内存泄漏的问题，是面向对象的数据结构。分了两个部分，头部与数据部分
- **IplImage**是从2001年OpenCV发布之后就一直存在，是C语言风格的数据结构，需要开发者自己分配与管理内存，对大的程序使用它容易导致内存泄漏问题

Mat对象构造函数与常用方法

Mat ()
Mat (int rows, int cols, int type)
Mat (Size size, int type)
Mat (int rows, int cols, int type, const Scalar &s)
Mat (Size size, int type, const Scalar &s)
Mat (int ndims, const int *sizes, int type)
Mat (int ndims, const int *sizes, int type, const Scalar &s)

常用方法：

void copyTo(Mat mat)
void convertTo(Mat dst, int type)
Mat clone()
int channels()
int depth()
bool empty();
uchar* ptr(i=0)

Mat对象使用

- 部分复制：一般情况下只会复制Mat对象的头和指针部分，不会复制数据部分

```
Mat A= imread(imgFilePath);
Mat B(A)  // 只复制
```

- 完全复制：如果想把Mat对象的头部和数据部分一起复制，可以通过如下两个API实现

```
Mat F = A.clone(); 或 Mat G; A.copyTo(G);
```

Mat对象使用-四个要点

- 输出图像的内存是自动分配的
- 使用OpenCV的C++接口，不需要考虑内存分配问题
- 赋值操作和拷贝构造函数只会复制头部分
- 使用clone与copyTo两个函数实现数据完全复制

Mat对象创建

- cv::Mat::Mat构造函数

  Mat M(2,2,CV_8UC3, Scalar(0,0,255))

  其中前两个参数分别表示行(row)跟列(column)、第三个CV_8UC3中的8表示每个通道占8位、U表示无符号、C表示Char类型、3表示通道数目是3，第四个参数是向量表示初始化每个像素值是多少，向量长度对应通道数目一致

```
M =
 [0, 0, 255, 0, 0, 255;
  0, 0, 255, 0, 0, 255]
```

- 创建多维数组cv::Mat::create

```
int sz[3] = {2,2,2};
Mat  L(3,sz, CV_8UC1, Scalar::all(0));
```

- cv::Mat::create实现

```
Mat M;
M.create(4, 3, CV_8UC2);
M = Scalar(127,127);
cout << "M = " << endl << " " << M << endl << endl;
uchar* firstRow = M.ptr<uchar>(0);
printf("%d", *firstRow);
```

```
M =
 [127, 127, 127, 127, 127, 127;
 127, 127, 127, 127, 127, 127;
 127, 127, 127, 127, 127, 127;
 127, 127, 127, 127, 127, 127]

127
```

定义小数组

```
Mat C = (Mat_<double>(3,3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);
cout << "C = " << endl << " " << C << endl << endl;
```

```
C =
 [0, -1, 0;
  -1, 5, -1;
  0, -1, 0]
```

演示代码：

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;
int main(int argc, char** args) {
Mat image = imread("D:/test.jpg", IMREAD_GRAYSCALE);
if (image.empty()) {
cout << "could not find the image resource..." << std::endl;
return -1;
}
namedWindow("My Image", CV_WINDOW_AUTOSIZE);
imshow("My Image", image);

Mat M;
M.create(4, 3, CV_8UC2);
M = Scalar(127,127);
cout << "M = " << endl << " " << M << endl << endl;
uchar* firstRow = M.ptr<uchar>(0);
printf("%d\n", *firstRow);

Mat C = (Mat_<double>(3, 3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);
cout << "C = " << endl << " " << C << endl << endl;

waitKey(0);
return 0;
}
```

4.图像操作

- 读写图像
- 读写像素
- 修改像素值

读写图像

- **imread** 可以指定加载为灰度或者RGB图像
- **Imwrite** 保存图像文件，类型由扩展名决定

读写像素

- 读一个GRAY像素点的像素值（CV_8UC1）

```
Scalar intensity = img.at<uchar>(y, x);
或者 Scalar intensity = img.at<uchar>(Point(x, y));
```

- 读一个RGB像素点的像素值

```
Vec3f intensity = img.at<Vec3f>(y, x);
float blue = intensity.val[0];
float green = intensity.val[1];
float red = intensity.val[2];
```

修改像素值

- 灰度图像

```
img.at<uchar>(y, x) = 128;
```

- RGB三通道图像

```
img.at<Vec3b>(y,x)[0]=128; // blue
img.at<Vec3b>(y,x)[1]=128; // green
img.at<Vec3b>(y,x)[2]=128; // red
```

- 空白图像赋值

```
img = Scalar(0);
```

- ROI选择

```
Rect r(10, 10, 100, 100);
Mat smallImg = img(r);
```

Vec3b与Vec3F

- Vec3b对应三通道的顺序是blue、 green、 red的uchar类型数据。
- Vec3f对应三通道的float类型数据
- 把CV_8UC1转换到CV32F1实现如下：

```
src.convertTo(dst, CV_32F);
```

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;
int main(int argc, char** args) {
Mat image = imread("D:/test.jpg", IMREAD_COLOR);
if (image.empty()) {
cout << "could not find the image resource..." << std::endl;
return -1;
}

int height = image.rows;
int width = image.cols;
int channels = image.channels();
printf("height=%d width=%d channels=%d", height, width, channels);
for (int row = 0; row < height; row++) {
      for (int col = 0; col < width; col++) {
            if (channels == 3) {
                  image.at<Vec3b>(row, col)[0] = 0; // blue
                  image.at<Vec3b>(row, col)[1] = 0; // green
            }
       }
}

namedWindow("My Image", CV_WINDOW_AUTOSIZE);
imshow("My Image", image);
waitKey(0);
return 0;
}
```

5.图像混合

- 理论-线性混合操作
- 相关API (addWeighted)
- 代码演示

理论-线性混合操作

$$g(x) = (1-\alpha) f_0(x) + \alpha f_1(x)$$

其中 α 的取值范围为0~1之间

相关API (addWeighted)

```
void cv::addWeighted ( InputArray  src1,
                       double      alpha,
                       InputArray  src2,
                       double      beta,
                       double      gamma,
                       OutputArray dst,
                       int         dtype = -1
                     )
```

$$\texttt{dst}(I) = \texttt{saturate}(\texttt{src1}(I) * \texttt{alpha} + \texttt{src2}(I) * \texttt{beta} + \texttt{gamma})$$

参数1：输入图像Mat – src1
参数2：输入图像src1的alpha值
参数3：输入图像Mat – src2
参数4：输入图像src2的alpha值
参数5：gamma值
参数6：输出混合图像

注意点：两张图像的大小和类型必须一致才可以

```
Mat src1, src2, dest;
src1 = imread("D:/vcprojects/images/LinuxLogo.jpg");
src2 = imread("D:/vcprojects/images/win7logo.jpg");
if (!src1.data) {
		printf("could not load LinuxLogo image...\n");
		return -1;
}
if (!src2.data) {
		printf("could not load win7logo image...\n");
		return -1;
}
if (src1.rows == src2.rows && src1.cols == src2.cols) {
		double alpha = 0.5;
		namedWindow("line-blend", CV_WINDOW_AUTOSIZE);
		addWeighted(src1, (1 - alpha), src2, alpha, 0.0, dest);

		imshow("line-blend", dest);
		waitKey(0);
		return 0;
}
else {
		printf("image size is not same...\n");
		return -1;
}
```

6.调整图像亮度与对比度

- 理论
- 代码演示

理论

- **图像变换**可以看作如下：

- 像素变换 – 点操作
- 邻域操作 – 区域

调整图像亮度和对比度属于像素变换-点操作

$$g(i,j) = \alpha f(i,j) + \beta \text{其中} \alpha > 0, \beta \text{是增益变量}$$

重要的API

- Mat new_image = Mat::zeros(image.size(), image.type()); 创建一张跟原图像大小和类型一致的空白图像、像素值初始化为0
- saturate_cast<uchar>(value)确保值大小范围为0~255之间
- Mat.at<Vec3b>(y,x)[index]=value 给每个像素点每个通道赋值

实现代码

```
Mat input, output;
input = imread("D:/vcprojects/images/test1.png");
namedWindow("input-image", CV_WINDOW_AUTOSIZE);
imshow("input-image", input);

if (!input.data) {
    printf("could not load image...\n");
    return -1;
}

int height = input.rows;
int width = input.cols;
double alpha = 1.2;
double beta = 50;
output = Mat::zeros(input.size(), input.type());
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        output.at<Vec3b>(y, x)[0] = saturate_cast<uchar>(alpha * input.at<Vec3b>(y, x)[0] + beta); // blue
        output.at<Vec3b>(y, x)[1] = saturate_cast<uchar>(alpha * input.at<Vec3b>(y, x)[1] + beta);  // green
        output.at<Vec3b>(y, x)[2] = saturate_cast<uchar>(alpha * input.at<Vec3b>(y, x)[2] + beta);  // red
    }
}

namedWindow("line-transform", CV_WINDOW_AUTOSIZE);
imshow("line-transform", output);
waitKey(0);

return 0;
```

7.绘制形状与文字

- 使用cv::Point与cv::Scalar
- 绘制线、矩形、园、椭圆等基本几何形状
- 随机生成与绘制文本
- 代码演示

使用cv::Point与cv::Scalar

- Point表示2D平面上一个点x,y

  Point p;

  p.x = 10;

  p.y = 8;

  or

  p = Pont(10,8);

- Scalar表示四个元素的向量

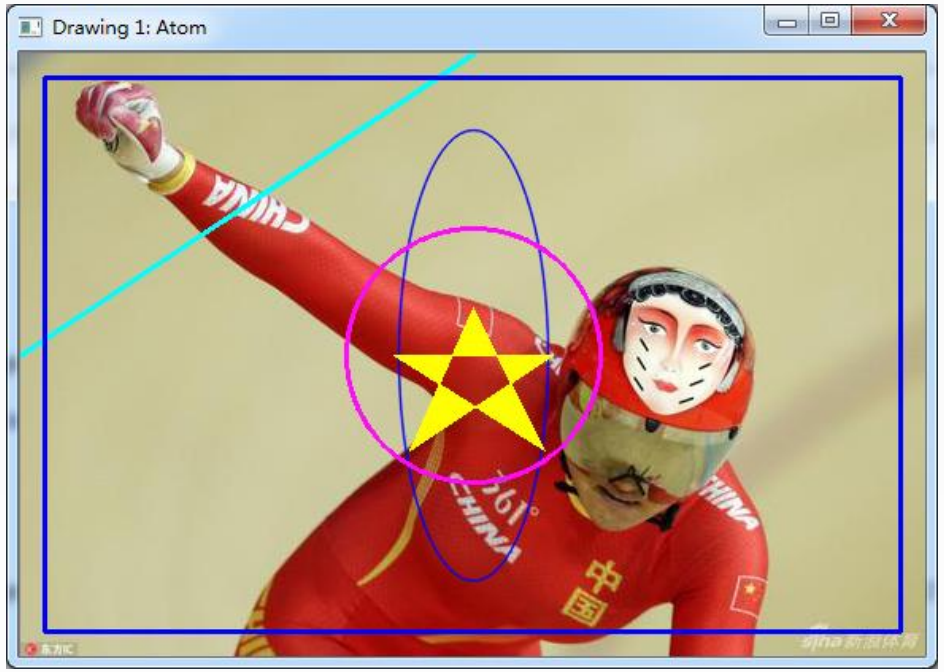
  Scalar(a, b, c);// a = blue, b = green, c = red表示RGB三个通道

绘制线、矩形、园、椭圆等基本几何形状

- 画线 cv::line （LINE_4\LINE_8\LINE_AA）
- 画椭圆cv::ellipse
- 画矩形cv::rectangle
- 画圆cv::circle
- 画填充cv::fillPoly

填充矩形

- 可以通过多边形填充来填充矩形



随机数生成cv::RNG

- 生成高斯随机数gaussian (double sigma)
- 生成正态分布随机数uniform (int a, int b)

绘制添加文字

- putText函数 中设置*fontFace(cv::HersheyFonts)*,

- fontFace, CV_FONT_HERSHEY_PLAIN

- fontScale , 1.0, 2.0~ 8.0

演示代码

```
int drawRandomLines(Mat image) {
    RNG rng(0xffffff);
    Point pt1, pt2;
    for (int i = 0; i < 100000; i++) {
        pt1.x = rng.uniform(0, image.cols);
        pt2.x = rng.uniform(0, image.cols);
        pt1.y = rng.uniform(0, image.rows);
        pt2.y = rng.uniform(0, image.rows);
        int r = rng.uniform(0, 255);
        int g = rng.uniform(0, 255);
        int b = rng.uniform(0, 255);
        line(image, pt1, pt2, Scalar(b, g, r), 1, LINE_8);
        putText(image, "Open CV Core Tutorial", Point(image.cols / 2-200, image.rows / 2),
            CV_FONT_HERSHEY_COMPLEX, 1.0, Scalar(0, 255, 0), 3, LINE_8);

        imshow(WINTITLE, image);
        if (waitKey(10) >= 0)
        {
            return -1;
        }
    }
    return 0;
}
```

8.模糊图像一

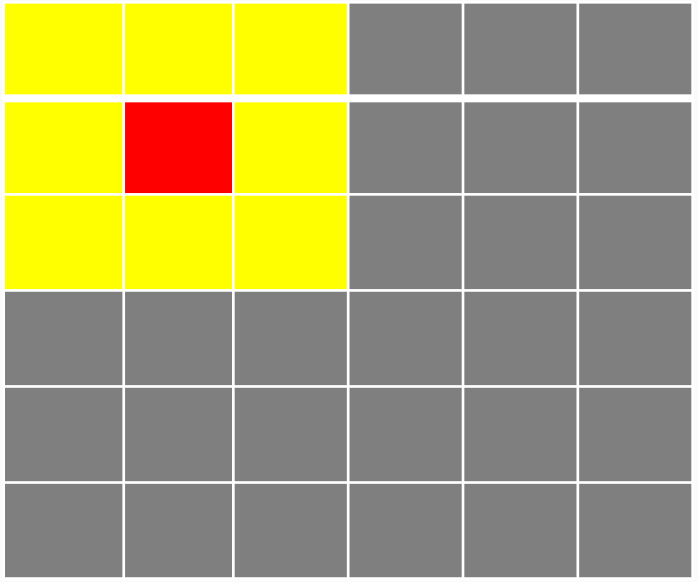
- 模糊原理
- 代码演示

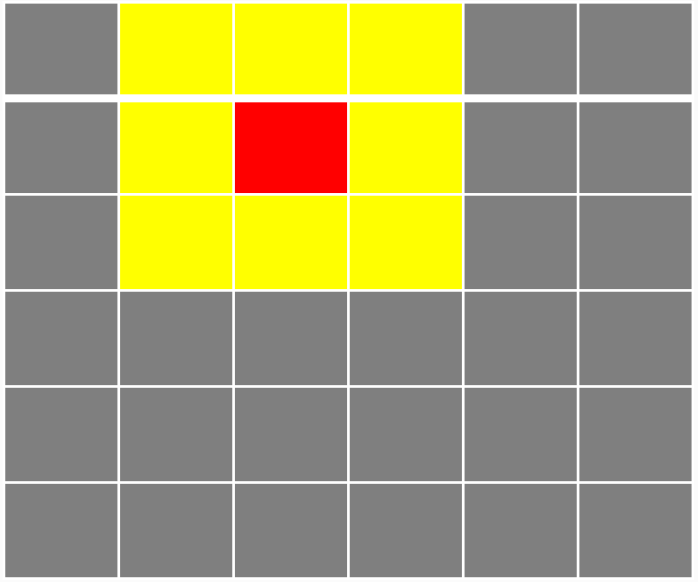
模糊原理

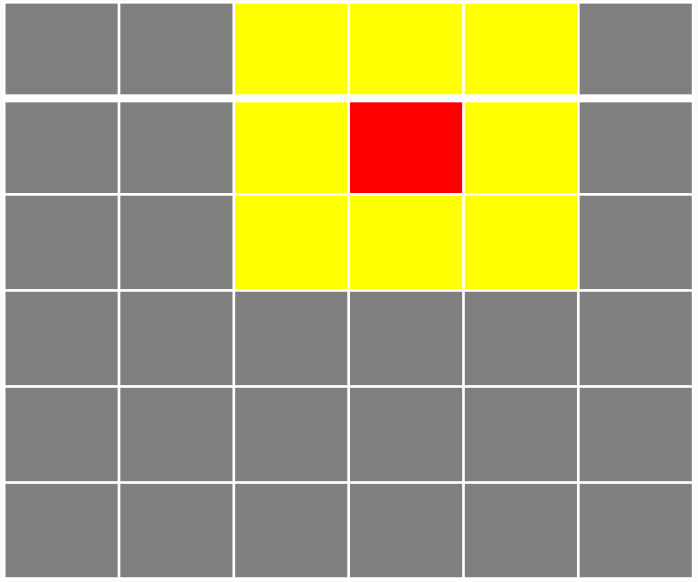
- Smooth/Blur 是图像处理中最简单和常用的操作之一
- 使用该操作的原因之一就为了给图像预处理时候减低噪声
- 使用Smooth/Blur操作其背后是数学的卷积计算

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

- 通常这些卷积算子计算都是线性操作，所以又叫线性滤波







假设有6x6的图像像素点矩阵。

卷积过程：6x6上面是个3x3的窗口，从左向右，从上向下移动，黄色的每个像个像素点值之和取平均值赋给中心红色像素作为它卷积处理之后新的像素值。每次移动一个像素格。

模糊原理

- 归一化盒子滤波（均值滤波）

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

- 高斯滤波

$$G_0(x, y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}}$$

相关API

- 均值模糊

  - blur(Mat src, Mat dst, Size(xradius, yradius), Point(-1,-1));

$$\texttt{dst}(x,y) = \sum_{\substack{0 \le x' < \texttt{kernel.cols},\\ 0 \le y' < \texttt{kernel.rows}}} \texttt{kernel}(x',y') * \texttt{src}(x + x' - \texttt{anchor.x}, y + y' - \texttt{anchor.y})$$

- 高斯模糊

  - GaussianBlur(Mat src, Mat dst, Size(11, 11), sigmax, sigmay);

其中Size（x, y）, x, y 必须是正数而且是奇数

演示代码

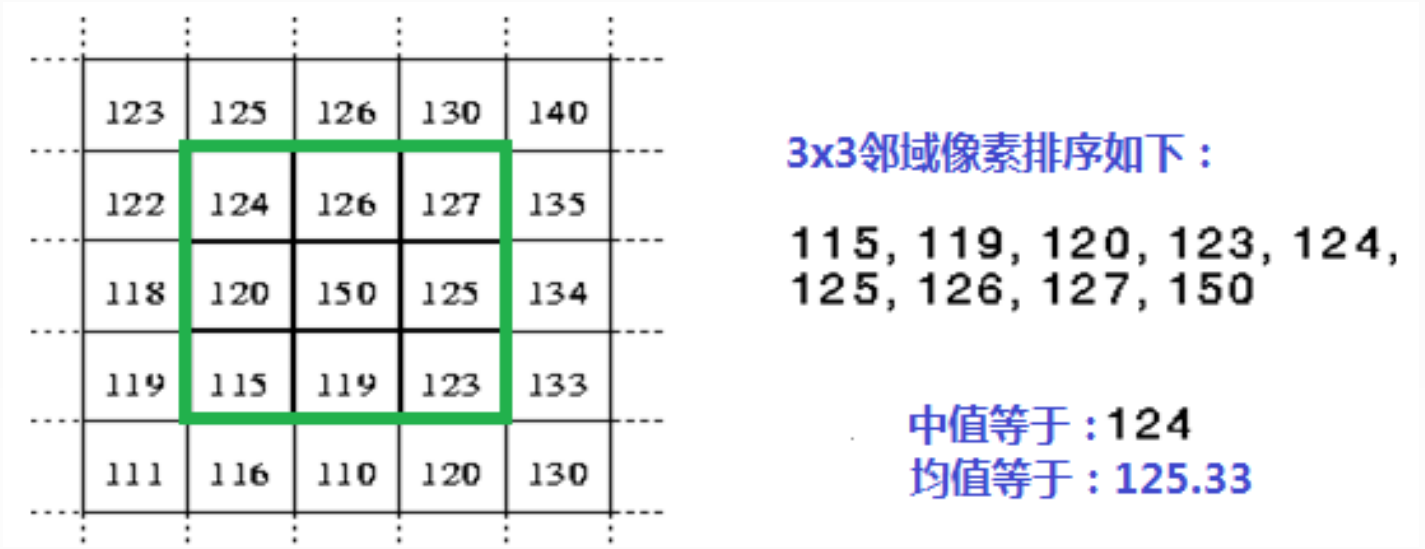
```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace cv;

int main(int argc, char** argv) {
    Mat src, dest;
    src = imread("D:/vcprojects/images/test.png");
    if (!src.data) {
        printf("could not load LinuxLogo image...\n");
        return -1;
    }
    char source_title[] = "sourceImage";
    char dest_title[] = "resultImage";

    namedWindow(source_title, CV_WINDOW_AUTOSIZE);
    namedWindow(dest_title, CV_WINDOW_AUTOSIZE);
    // blur(src, dest, Size(15, 15), Point(-1, -1));

    GaussianBlur(src, dest, Size(11, 11), 5, 5);

    imshow(source_title, src);
    imshow(dest_title, dest);
    waitKey(0);
    return 0;
}
```

9.图像模糊二

- 中值滤波
- 双边滤波
- 代码演示

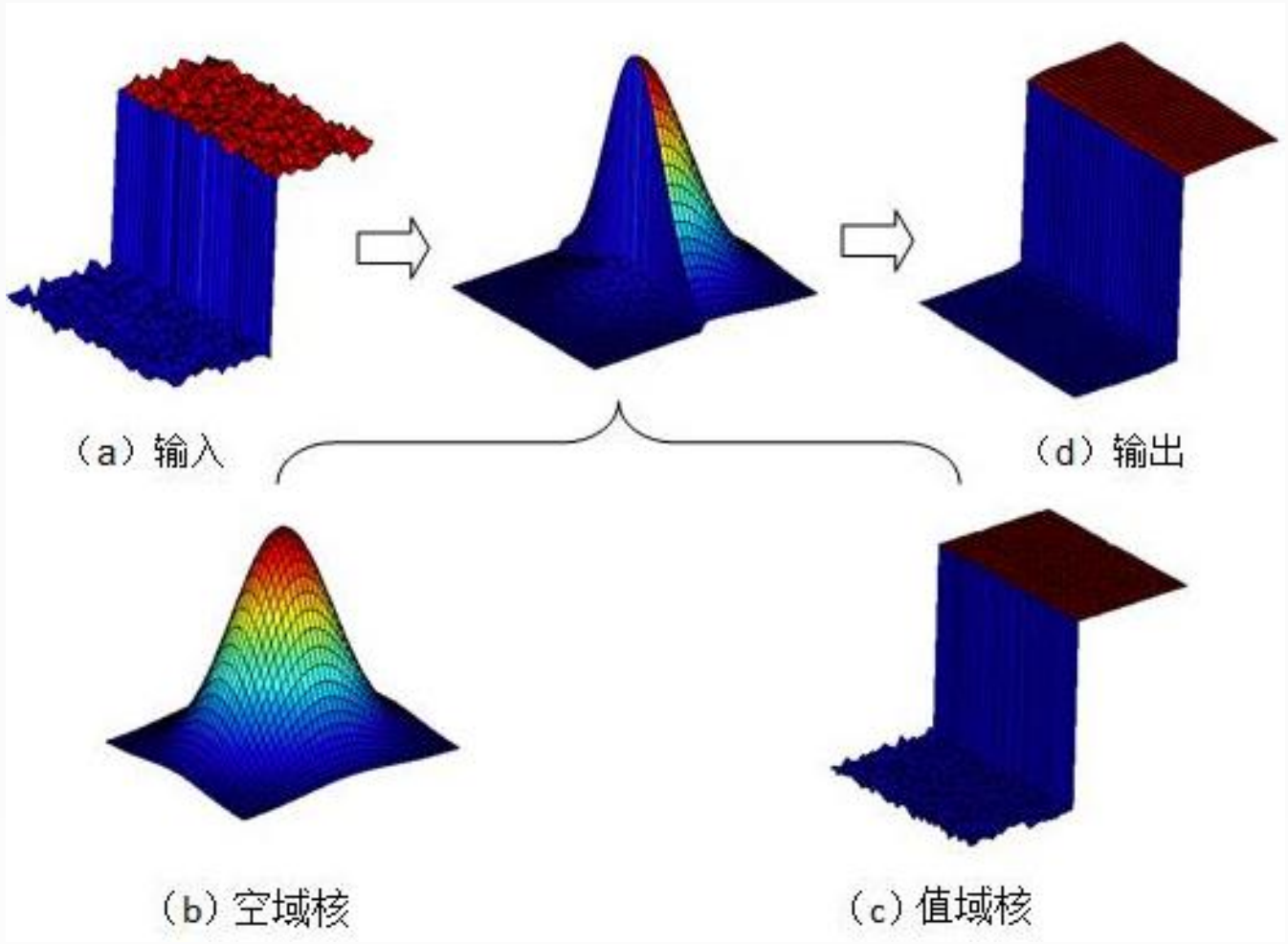
中值滤波

- 统计排序滤波器
- 中值对椒盐噪声有很好的抑制作用



双边滤波

- 均值模糊无法克服边缘像素信息丢失缺陷。原因是均值滤波是基于平均权重
- 高斯模糊部分克服了该缺陷，但是无法完全避免，因为没有考虑像素值的不同
- 高斯双边模糊 – 是边缘保留的滤波方法，避免了边缘信息丢失，保留了图像轮廓不变


（a）输入
（d）输出
（b）空域核
（c）值域核

相关API

- 中值模糊medianBlur（Mat src, Mat dest, ksize）
- 双边模糊bilateralFilter(src, dest, d=15, 150, 3);

- 15 –计算的半径，半径之内的像数都会被纳入计算，如果提供-1 则根据sigma space参数取值
- 150 – sigma color 决定多少差值之内的像素会被计算
- 3 – sigma space 如果d的值大于0则声明无效，否则根据它来计算d值

中值模糊的ksize大小必须是大于1而且必须是奇数。

演示代码

```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace cv;

int main(int argc, char** argv) {
    Mat src, dest;
    src = imread("D:/vcprojects/images/cvtest.png");
    if (!src.data) {
        printf("could not load LinuxLogo image...\n");
        return -1;
    }

    char INPUT_WIN[] = "Source Image";
    char OUTPUT_WIN[] = "Filted out Image";
    namedWindow(INPUT_WIN, CV_WINDOW_AUTOSIZE);
    namedWindow(OUTPUT_WIN, CV_WINDOW_AUTOSIZE);

    // filter image out
    // medianBlur(src, dest, 3);
    bilateralFilter(src, dest, 15, 150, 10);

    imshow(INPUT_WIN, src);
    imshow(OUTPUT_WIN, dest);
    waitKey(0);
    return 0;

}
```

10.膨胀与腐蚀

- 腐蚀
- 膨胀
- 代码演示

形态学操作(morphology operators)-膨胀

- 图像形态学操作 – 基于形状的一系列图像处理操作的合集，主要是基于集合论基础上的形态学数学
- 形态学有四个基本操作：腐蚀、膨胀、开、闭
- 膨胀与腐蚀是图像处理中最常用的形态学操作手段

形态学操作-膨胀

- 跟卷积操作类似，假设有图像A和结构元素B，结构元素B在A上面移动，其中B定义其中心为锚点，计算B覆盖下A的最大像素值用来替换锚点的像素，其中B作为结构体可以是任意形状

形态学操作-腐蚀

- 腐蚀跟膨胀操作的过程类似，唯一不同的是以最小值替换锚点重叠下图像的像素值

相关API

- getStructuringElement(int shape, Size ksize, Point anchor)

- 形状 (MORPH_RECT \MORPH_CROSS \MORPH_ELLIPSE)
- 大小
- 锚点 默认是Point(-1, -1)意思就是中心像素

- dilate(src, dst, kernel)

$$\texttt{dst}(x,y) = \max_{(x',y'):\, \texttt{element}(x',y') \neq 0} \texttt{src}(x+x', y+y')$$

- erode(src, dst, kernel)

$$\texttt{dst}(x,y) = \min_{(x',y'):\, \texttt{element}(x',y') \neq 0} \texttt{src}(x+x', y+y')$$

动态调整结构元素大小

- TrackBar – createTrackbar(const String & trackbarname, const String winName, int* value, int count, Trackbarcallback func, void* userdata=0)

其中最中要的是 callback 函数功能。如果设置为NULL就是说只有值update，但是不会调用callback的函数。

代码演示

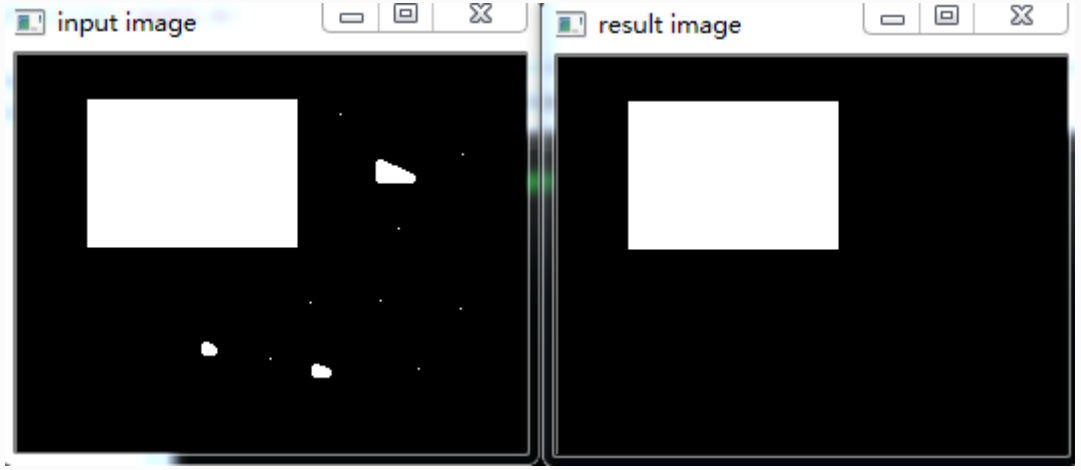
```
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace cv;

int main(int argc, char** argv) {
    Mat src, dest;
    src = imread("D:/vcprojects/images/cat.jpg");
    if (!src.data) {
        printf("could not load LinuxLogo image...\n");
        return -1;
    }

    char INPUT_WIN[] = "input image";
    char OUTPUT_WIN[] = "output image";
    namedWindow(INPUT_WIN, CV_WINDOW_AUTOSIZE);
    namedWindow(OUTPUT_WIN, CV_WINDOW_AUTOSIZE);

    // dilate
    Mat kernel = getStructuringElement(MORPH_RECT, Size(5, 5), Point(-1, -1));
    // dilate(src, dest, kernel);

    // erosion
    erode(src, dest, kernel);

    imshow(INPUT_WIN, src);
    imshow(OUTPUT_WIN, dest);
    waitKey(0);
    return 0;
}
```

11.形态学操作

- 开操作- open
- 闭操作- close
- 形态学梯度- Morphological Gradient
- 顶帽 – top hat
- 黑帽 – black hat

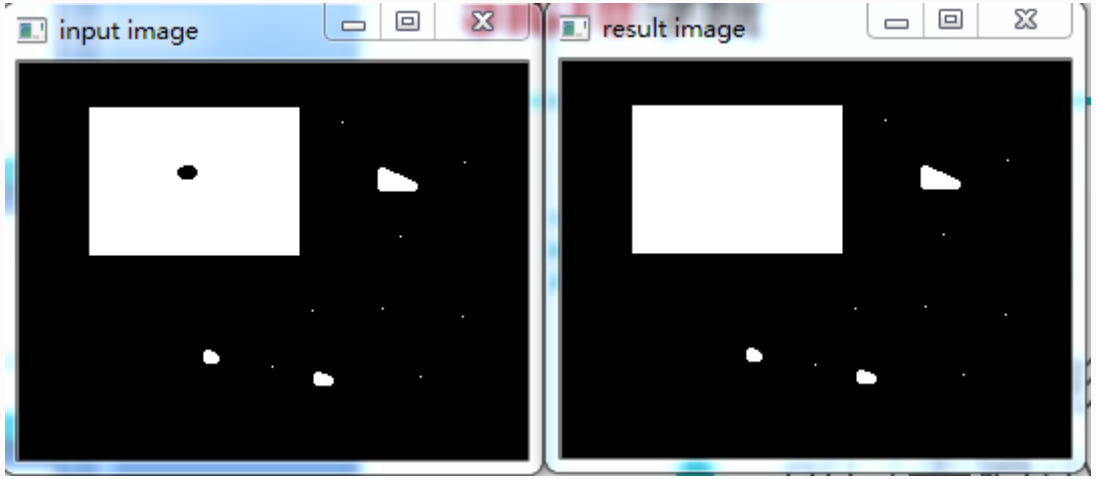
开操作- open

- 先腐蚀后膨胀 $dst = open(src, element) = dilate(erode(src, element))$
- 可以去掉小的对象，假设对象是前景色，背景是黑色



闭操作-close

- 先膨胀后腐蚀（bin2） $dst = close(src, element) = erode(dilate(src, element))$
- 可以填充小的洞（fill hole），假设对象是前景色，背景是黑色



形态学梯度- Morphological Gradient

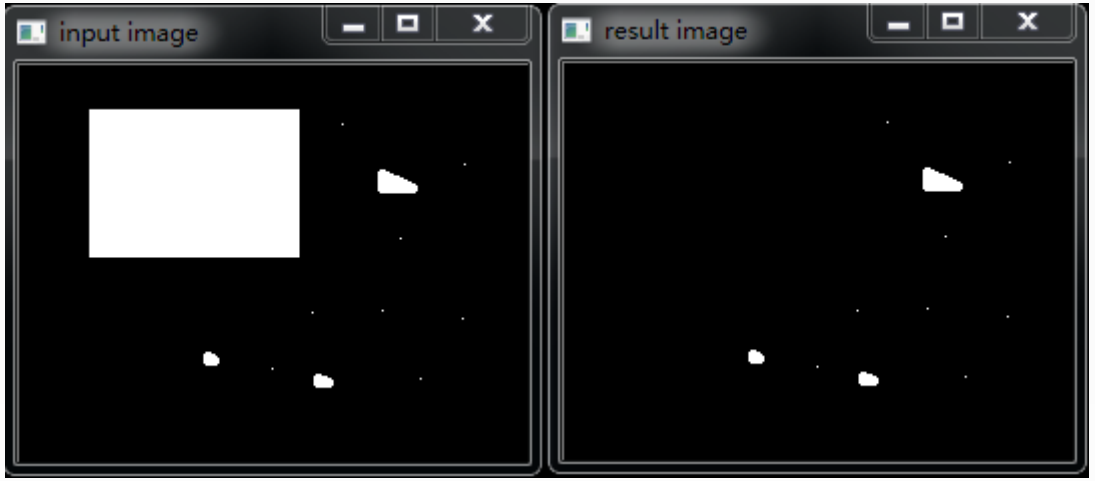
- 膨胀减去腐蚀

$$dst = morph_{grad}(src, element) = dilate(src, element) - erode(src, element)$$

- 又称为基本梯度（其它还包括-内部梯度、方向梯度）

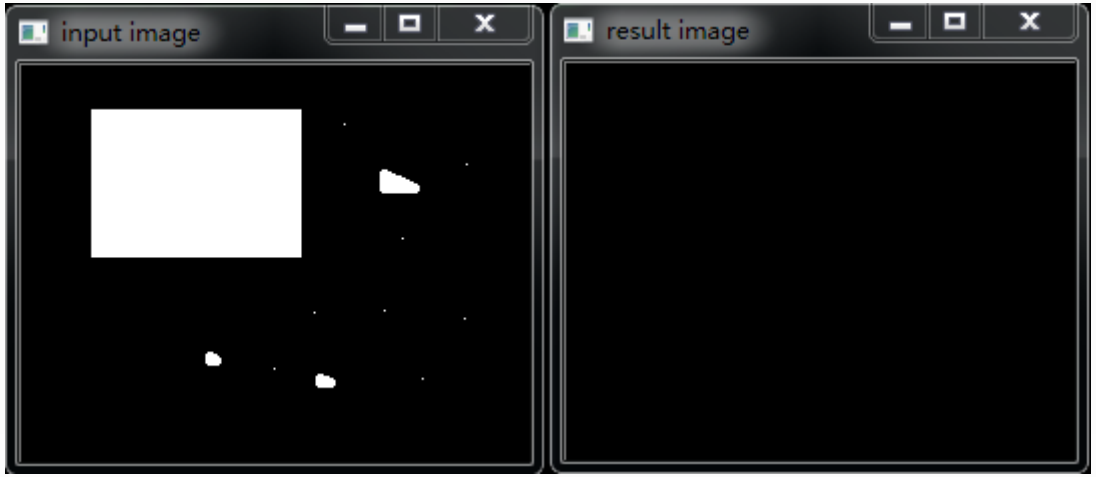
顶帽 – top hat

- 顶帽 是原图像与开操作之间的差值图像



黑帽

- 黑帽是闭操作图像与源图像的差值图像



相关API

- morphologyEx(src, dest, CV_MOP_BLACKHAT, kernel);

- Mat src – 输入图像
- Mat dest – 输出结果
- int OPT – CV_MOP_OPEN/ CV_MOP_CLOSE/ CV_MOP_GRADIENT / CV_MOP_TOPHAT/ CV_MOP_BLACKHAT 形态学操作类型
- Mat kernel 结构元素
- int Iteration 迭代次数，默认是1

演示代码

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

using namespace cv;
int main(int argc, char** argv) {
    Mat src, dest;
    src = imread("D:/vcprojects/images/bintest.png");
    if (!src.data) {
        printf("could not load image...\n");
    }

    char INPUT_WIN[] = "input image";
    char OUTPUT_WIN[] = "result image";
    namedWindow(INPUT_WIN, CV_WINDOW_AUTOSIZE);
    namedWindow(OUTPUT_WIN, CV_WINDOW_AUTOSIZE);

    Mat kernel = getStructuringElement(MORPH_RECT, Size(11, 11), Point(-1, -1));

    morphologyEx(src, dest, CV_MOP_BLACKHAT, kernel);

    imshow(INPUT_WIN, src);
    imshow(OUTPUT_WIN, dest);
    waitKey(0);
    return 0;
}
```

12.形态学操作应用-提取水平与垂直线

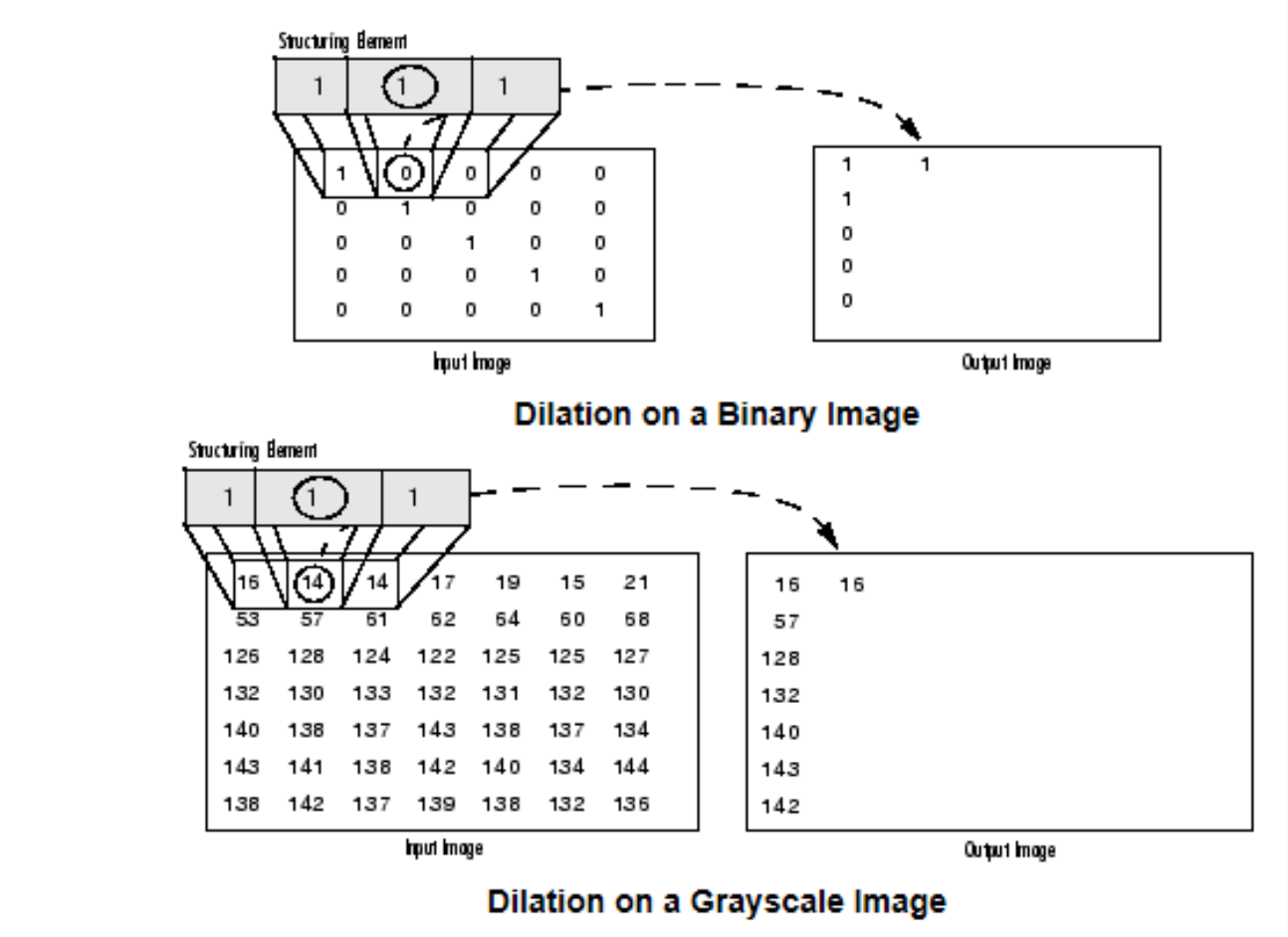
- 原理方法
- 实现步骤
- 代码演示

原理方法

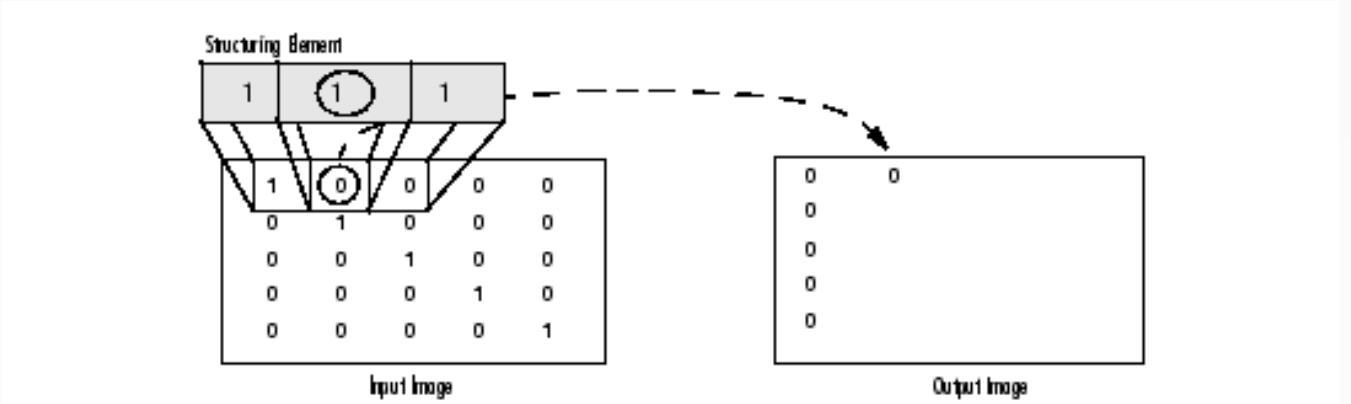
图像形态学操作时候，可以通过自定义的结构元素实现结构元素对输入图像一些对象敏感、另外一些对象不敏感，这样就会让敏感的对象改变而不敏感的对象保留输出。通过使用两个最基本的形态学操作 – **膨胀**与**腐蚀**，使用不同的结构元素实现对输入图像的操作、得到想要的结果。

- 膨胀，输出的像素值是结构元素覆盖下输入图像的最大像素值
- 腐蚀，输出的像素值是结构元素覆盖下输入图像的最小像素值

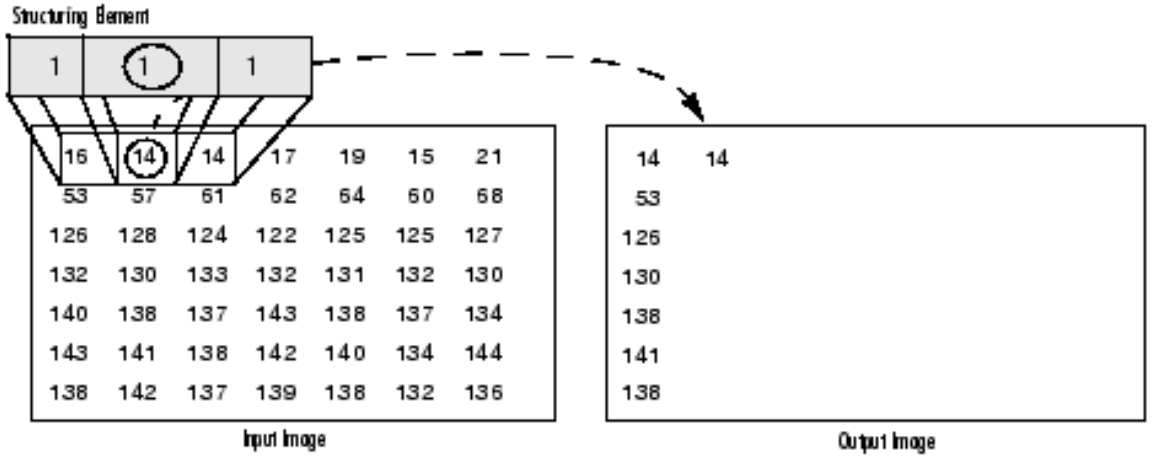
二值图像与灰度图像上的膨胀操作



二值图像与灰度图像上的腐蚀操作



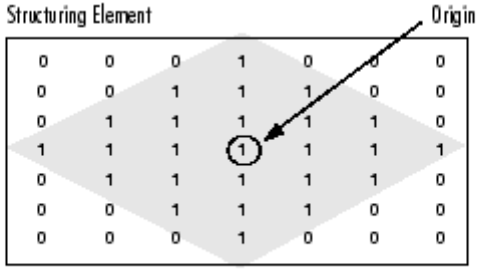
Erosion on a Binary Image



Erosion on a Grayscale Image

结构元素

- 上述膨胀与腐蚀过程可以使用任意的结构元素
- 常见的形状：矩形、园、直线、磁盘形状、砖石形状等各种自定义形状。

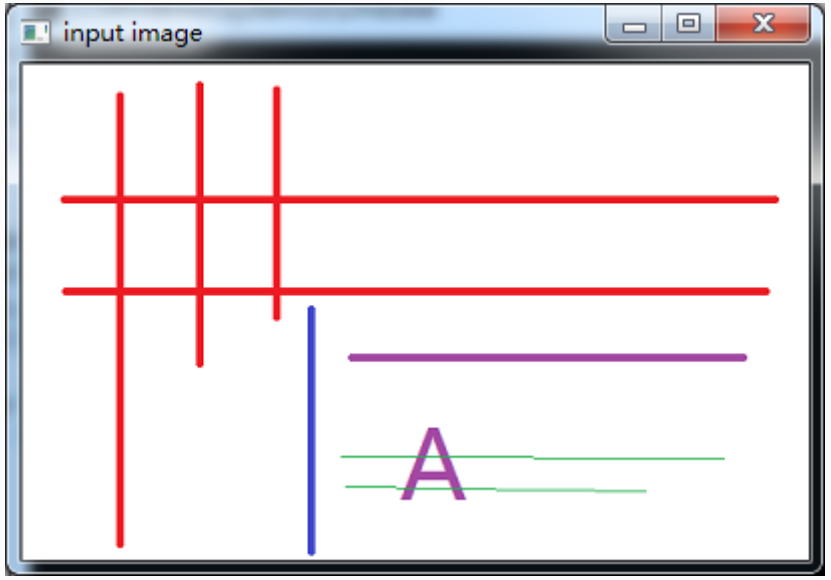


A Diamond-Shaped Structuring Element and its Origin

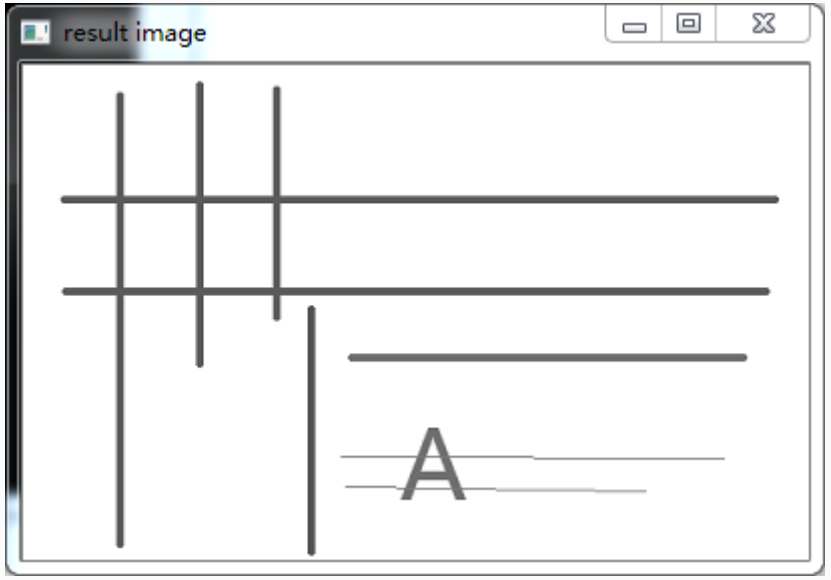
提取步骤

- 输入图像彩色图像 imread
- 转换为灰度图像 – cvtColor
- 转换为二值图像 – adaptiveThreshold
- 定义结构元素
- 开操作 （腐蚀+膨胀）提取 水平与垂直线

代码实现-第一步输入彩色图像 imread



转换为灰度图像 – cvtColor




```
Mat gray;
if (src.channels() == 3) {
    cvtColor(src, gray, CV_BGR2GRAY);
} else {
    gray = src;
}
imshow(OUTPUT_WIN, gray);
```

转换为二值图像 – adaptiveThreshold

- adaptiveThreshold(

```
Mat src, // 输入的灰度图像
Mat dest, // 二值图像
double maxValue, // 二值图像最大值
int adaptiveMethod // 自适应方法，只能其中之一 –
                   // ADAPTIVE_THRESH_MEAN_C ， ADAPTIVE_THRESH_GAUSSIAN_C
int thresholdType,// 阈值类型
int blockSize, // 块大小
double C // 常量C 可以是正数，0，负数
)
```

转换为二值图像 – adaptiveThreshold

- **THRESH_BINARY**

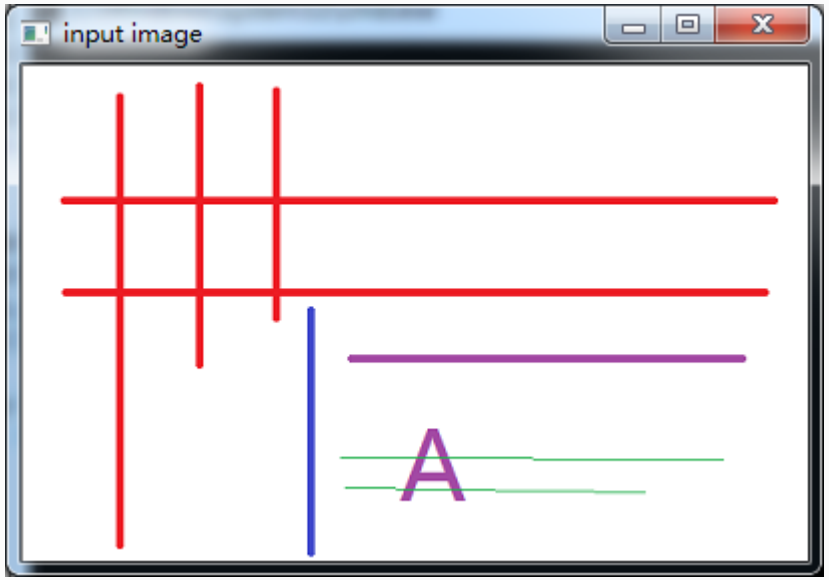
$$dst(x,y) = \begin{cases} \texttt{maxValue} & \text{if } src(x,y) > T(x,y) \\ 0 & \text{otherwise} \end{cases}$$

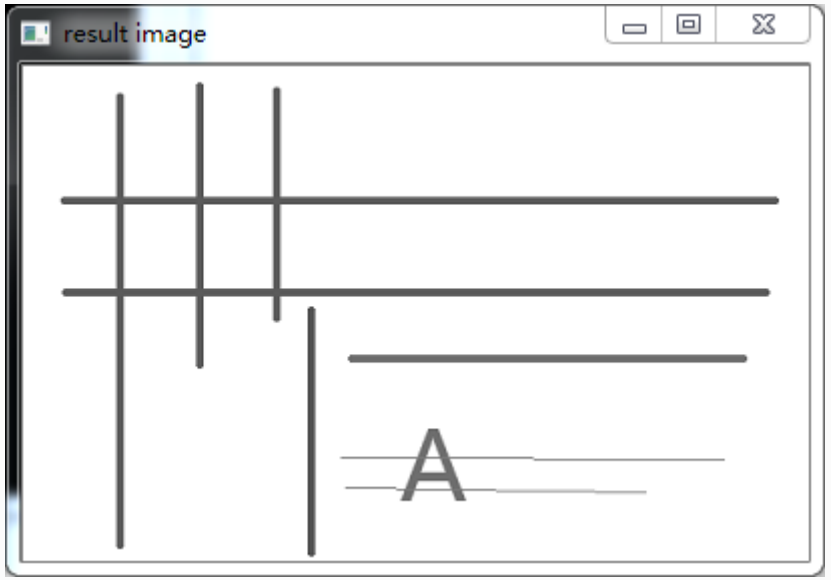
- **THRESH_BINARY_INV**

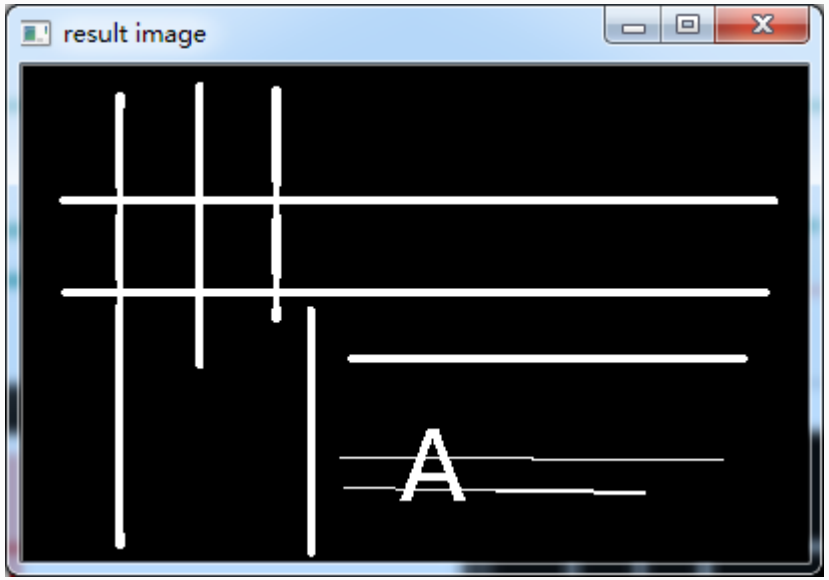
$$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > T(x,y) \\ \texttt{maxValue} & \text{otherwise} \end{cases}$$

阈值T = *sum*(blocksize X blockSize的像素平均值) - 常量C

转换为二值图像 – adaptiveThreshold



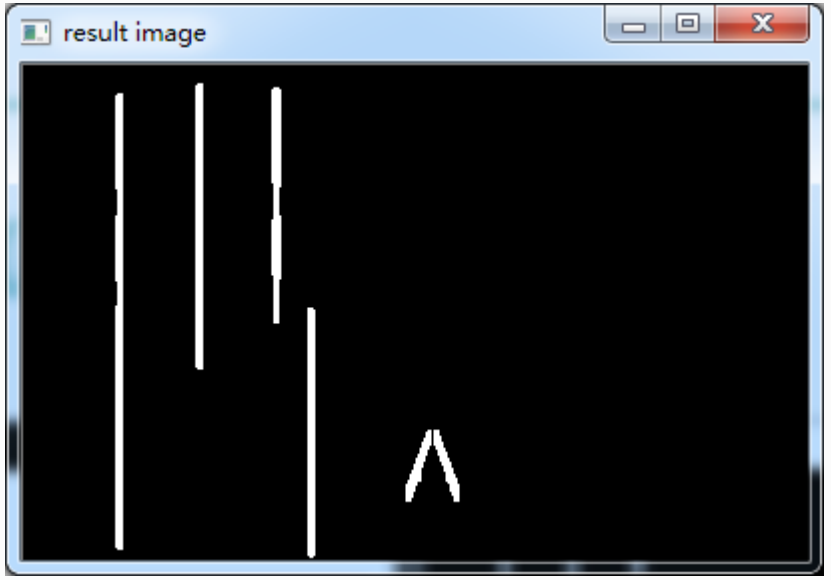


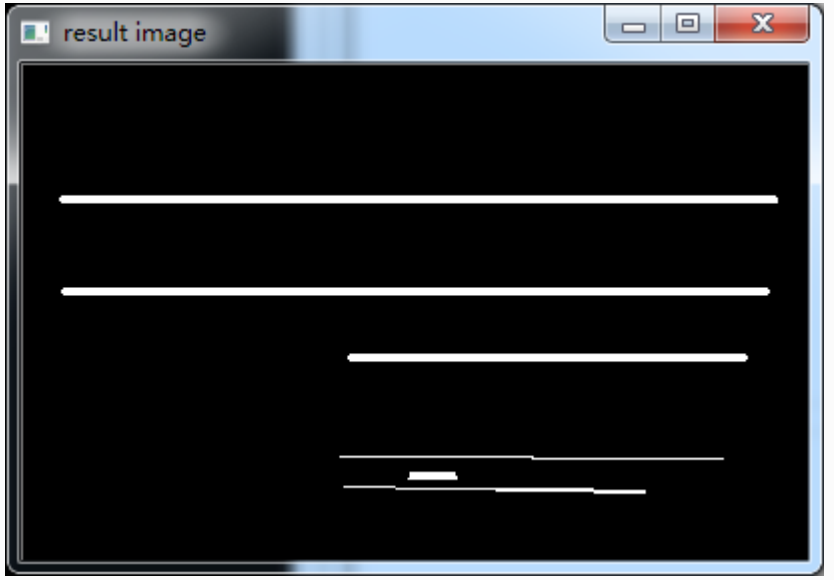


定义结构元素

- 一个像素宽的水平线 - 水平长度 width/30
- 一个像素宽的垂直线 – 垂直长度 height/30

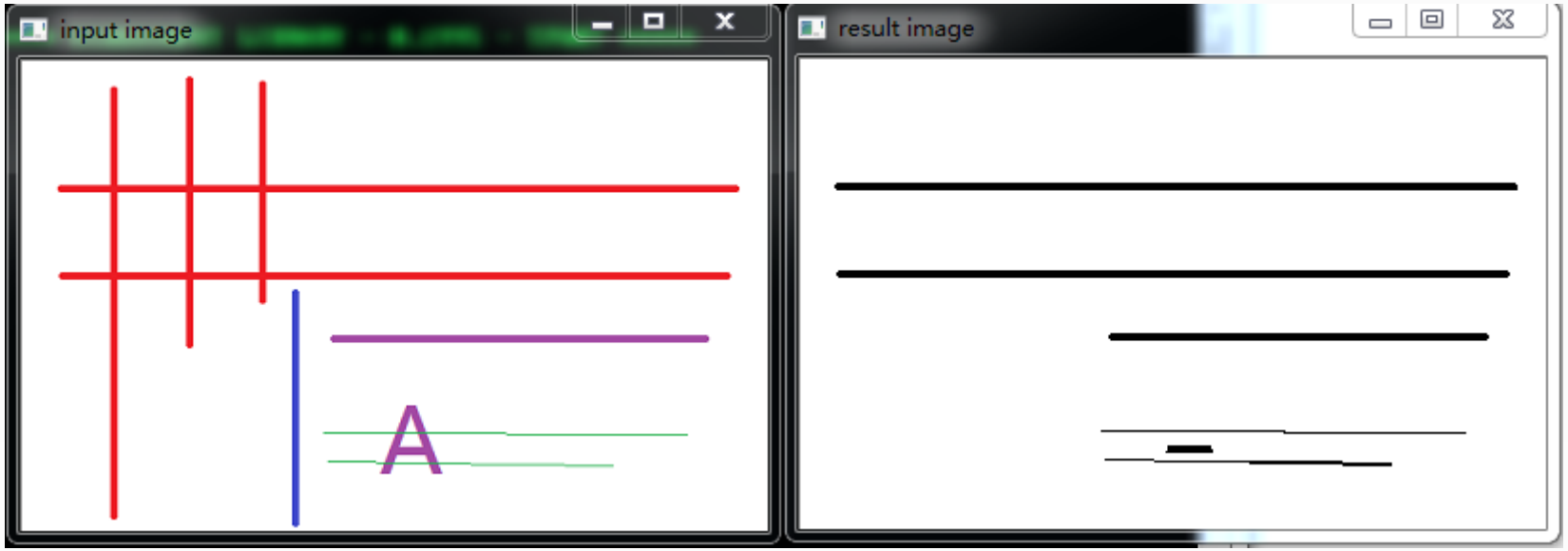
开操作(腐蚀+膨胀)-检测





后处理

- bitwise_not（Mat bin, Mat dst）像素取反操作，255 – SrcPixel
- 模糊（blur）



代码实现

```
Mat src, dest;
src = imread("D:/vcprojects/images/bin1.png");
if (!src.data) {
    printf("could not load image...\n");
    return -1;
}

char INPUT_WIN[] = "input image";
char OUTPUT_WIN[] = "result image";
namedWindow(INPUT_WIN, CV_WINDOW_AUTOSIZE);
imshow(INPUT_WIN, src);

// conver to gray image
Mat gray;
if (src.channels() == 3) {
    cvtColor(src, gray, CV_BGR2GRAY);
} else {
    gray = src;
}

// convert to binary image
adaptiveThreshold(~gray, dest, 255, ADAPTIVE_THRESH_MEAN_C, THRESH_BINARY, 15, -2);

// create custom structure
int xsize = dest.cols / 30;
int ysize = dest.rows / 30;
Mat horline = getStructuringElement(MORPH_RECT, Size(xsize, 1), Point(-1, -1));
Mat vecline = getStructuringElement(MORPH_RECT, Size(1, ysize), Point(-1, -1));

// open operation - extract horizle lines
Mat hbin;
erode(dest, hbin, horline);
dilate(hbin, dest, horline);
imshow(OUTPUT_WIN, dest);
```

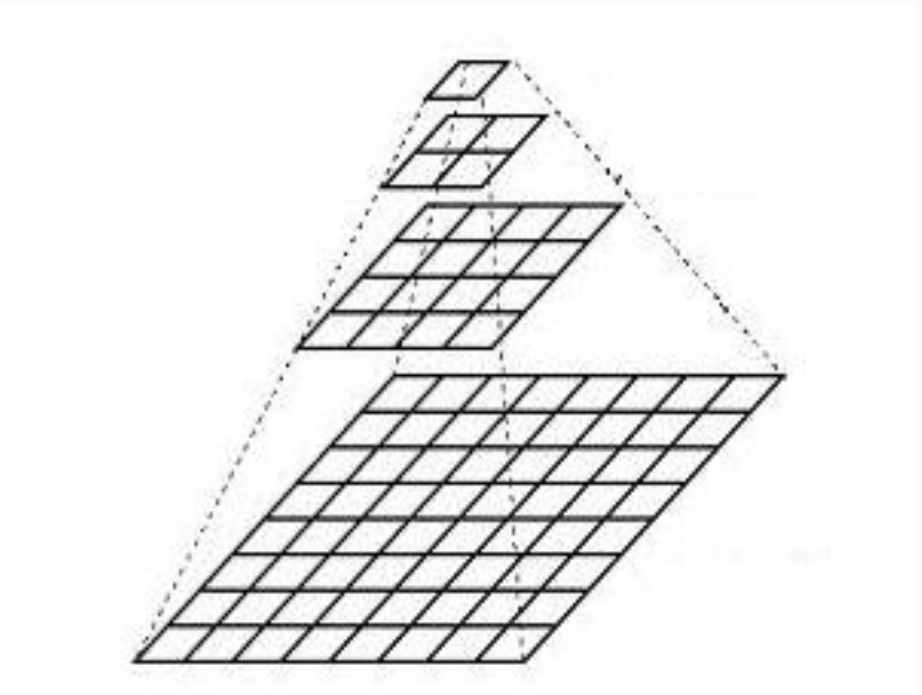
13.图像上采样和降采样

- 图像金字塔概念
- 采样API
- 代码演示

图像金字塔概念

1. 我们在图像处理中常常会调整图像大小，最常见的就是放大(zoom in)和缩小（zoom out），尽管几何变换也可以实现图像放大和缩小，但是这里我们介绍图像金字塔
2. 一个图像金字塔式一系列的图像组成，最底下一张是图像尺寸最大，最上方的图像尺寸最小，从空间上从上向下看就想一个古代的金字塔。

图像金字塔概念



图像金字塔概念

- 高斯金子塔 – 用来对图像进行降采样
- 拉普拉斯金字塔 – 用来重建一张图片根据它的上层降采样图片

图像金字塔概念 – 高斯金字塔

- 高斯金子塔是从底向上，逐层降采样得到。
- 降采样之后图像大小是原图像MxN的M/2 x N/2 ,就是对原图像删除偶数行与列，即得到降采样之后上一层的图片。
- 高斯金子塔的生成过程分为两步：
  - 对当前层进行高斯模糊
  - 删除当前层的偶数行与列

  即可得到上一层的图像，这样上一层跟下一层相比，都只有它的1/4大小。

$$\frac{1}{16}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

高斯不同(Difference of Gaussian-DOG)

- 定义：就是把同一张图像在不同的参数下做高斯模糊之后的结果相减，得到的输出图像。称为高斯不同(DOG)
- 高斯不同是图像的内在特征，在灰度图像增强、角点检测中经常用到。

采样相关API

- 上采样(cv::pyrUp) – zoom in 放大
- 降采样 (cv::pyrDown) – zoom out 缩小

pyrUp(Mat src, Mat dst, Size(src.cols*2, src.rows*2))
生成的图像是原图在宽与高各放大两倍
pyrDown(Mat src, Mat dst, Size(src.cols/2, src.rows/2))
生成的图像是原图在宽与高各缩小1/2

演示代码

```
Mat src, dest;
src = imread("D:/vcprojects/images/cat.jpg");
if (!src.data) {
    printf("could not load image...");
    return -1;
}

char INPUT_WIN[] = "input image";
char OUTPUT_WIN[] = "show result";
namedWindow(INPUT_WIN, CV_WINDOW_AUTOSIZE);
namedWindow(OUTPUT_WIN, CV_WINDOW_AUTOSIZE);
imshow(INPUT_WIN, src);

// zoom out
// pyrDown(src, dest, Size(src.cols / 2, src.rows / 2));
// imshow(OUTPUT_WIN, dest);

// zoom in
pyrUp(src, dest, Size(src.cols * 2, src.rows * 2));
imshow(OUTPUT_WIN, dest);

waitKey(0);
return 0;
```