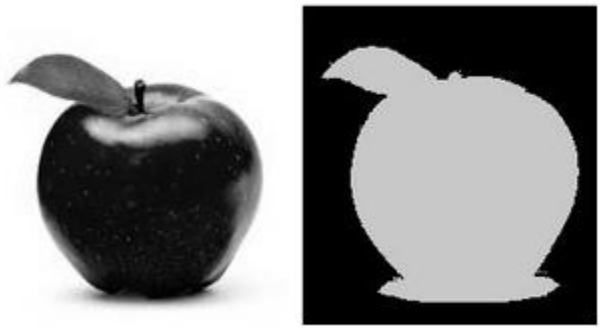
14.基本阈值操作

- 图像阈值
- 阈值类型
- 代码演示

为梦想增值！

图像阈值（threshold）

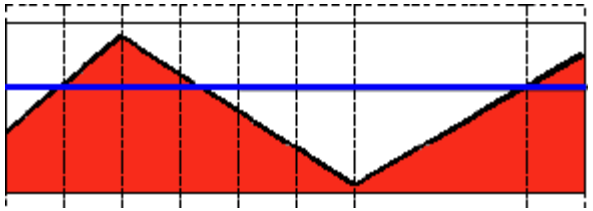
- **阈值** 是什么？简单点说是把图像分割的标尺，这个标尺是根据什么产生的，阈值产生算法？阈值类型。（Binary segmentation）



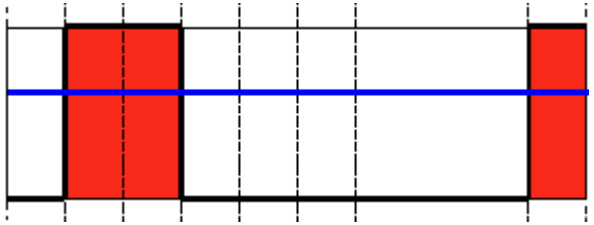
为梦想增值！

阈值类型—阈值二值化(threshold binary)

- 左下方的图表示图像像素点Src(x,y)值分布情况，蓝色水平线表示阈值

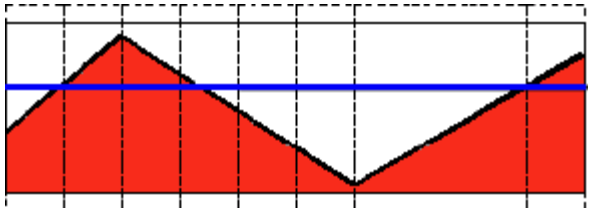


$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxVal} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

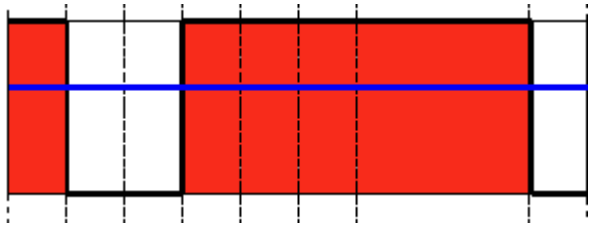


阈值类型—阈值反二值化(threshold binary Inverted)

- 左下方的图表示图像像素点Src(x,y)值分布情况，蓝色水平线表示阈值

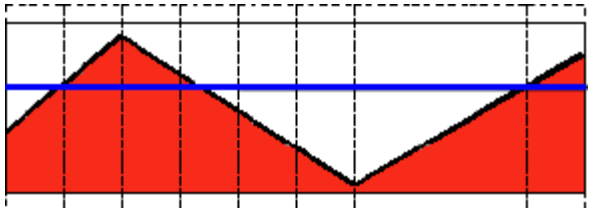


$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases}$$

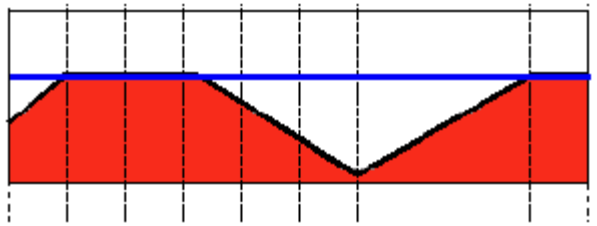


阈值类型—截断 (truncate)

- 左下方的图表示图像像素点Src(x,y)值分布情况，蓝色水平线表示阈值

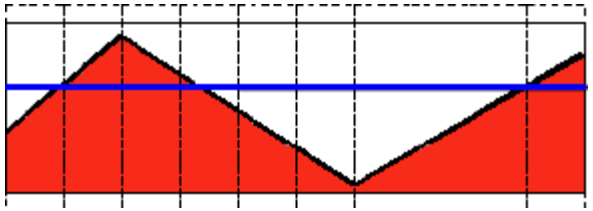


$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

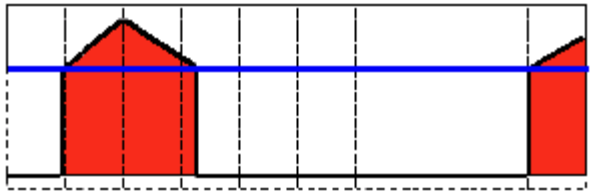


阈值类型—阈值取零 (threshold to zero)

- 左下方的图表示图像像素点Src(x,y)值分布情况，蓝色水平线表示阈值

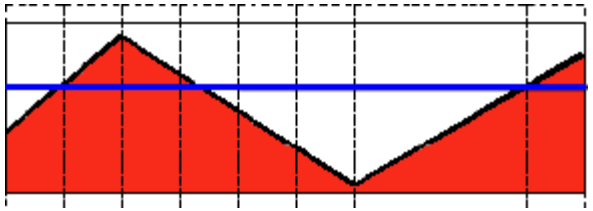


$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

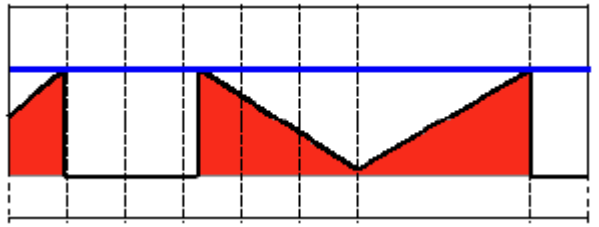


阈值类型—阈值反取零 (threshold to zero inverted)

- 左下方的图表示图像像素点Src(x,y)值分布情况，蓝色水平线表示阈值



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



Enumerator	
THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_MASK	
THRESH_OTSU	flag, use Otsu algorithm to choose the optimal threshold value
THRESH_TRIANGLE	flag, use Triangle algorithm to choose the optimal threshold value

演示代码

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

// apply threshold
createTrackbar(trackbar_value, OUTPUT_WIN, &threshold_value, maxvalue, threshold_demo);
threshold_demo(0, 0);

// loop for your end
for (;;) {
    int c;
    c = waitKey(20);
    if ((char)c == 27) { // ESC
        break;
    }
}
return 0;
```

为梦想增值！

15.自定义线性滤波

- 卷积概念
- 常见算子
- 自定义卷积模糊
- 代码演示

为梦想增值！

卷积概念

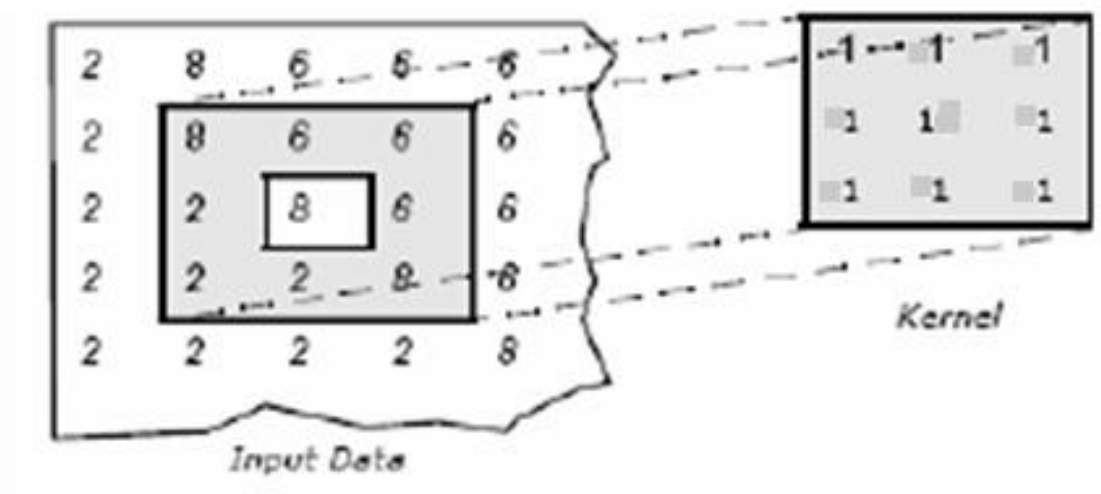
- **卷积**是图像处理中一个操作，是kernel在图像的每个像素上的操作。
- **Kernel**本质上一个固定大小的矩阵数组，其中心点称为锚点(anchor point)

1	-2	1
2	-4	2
1	-2	1

卷积如何工作

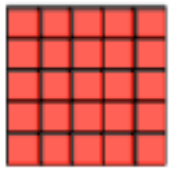
- 把kernel放到像素数组之上，求锚点周围覆盖的像素乘积之和（包括锚点），用来替换锚点覆盖下像素点值称为卷积处理。数学表达如下：

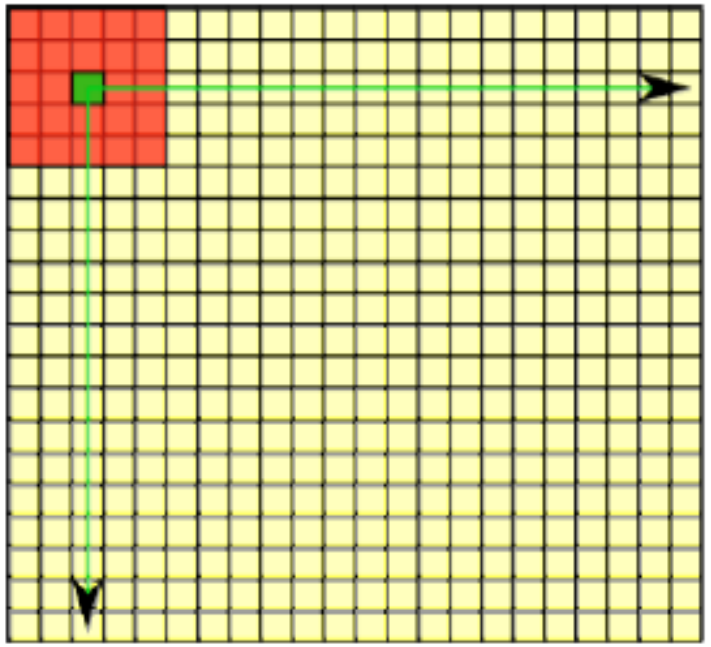
$$H(x,y)=\sum_{i=0}^{M_i-1}\sum_{j=0}^{M_j-1}I(x+i-a_i,y+j-a_j)K(i,j)$$

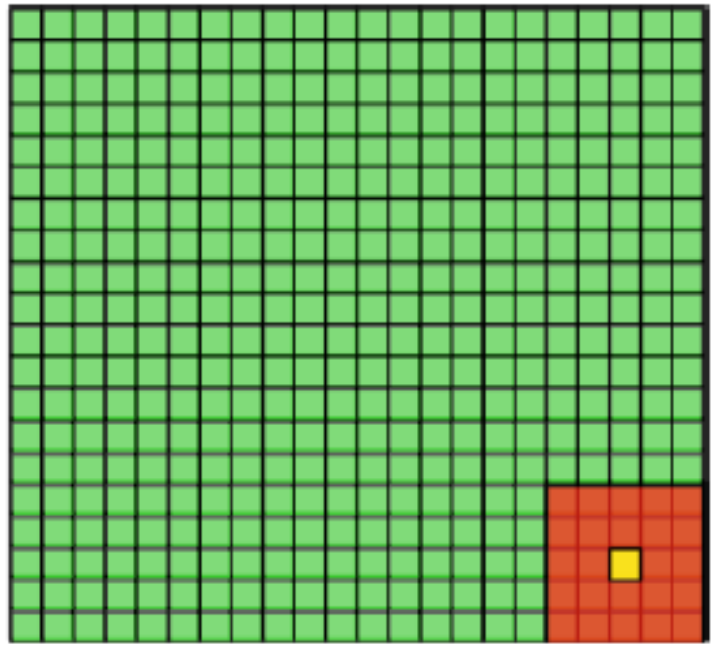


Sum = 8x1+6x1+6x1+2x1+8x1+6x1+2x1+2x1+8x1
New pixel = sum / (m*n)

为梦想增值！







为梦想增值！

常见算子

+1	0
0	-1

0	+1
-1	0

Robert算子

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Sobel算子

0	-1	0
-1	4	-1
0	-1	0

拉普拉斯算子

为梦想增值！

自定义卷积模糊

- filter2D方法filter2D(
Mat src, //输入图像
Mat dst, // 模糊图像
int depth, // 图像深度32/8
Mat kernel, // 卷积核/模板
Point anchor, // 锚点位置
double delta // 计算出来的像素+delta
)
其中 kernel是可以自定义的卷积核

$$K = \frac{1}{3 \cdot 3}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

演示代码

```
src = imread("D:/vcprojects/images/test.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);
namedWindow(OUTPUT_WIN, CV_WINDOW_AUTOSIZE);

imshow(INPUT_WIN, src);

int c = 0;
int index = 1;
while (true) {
    c = waitKey(500);
    if ((char)c == 27) {
        break;
    }
    ksize = 3 + 2 * (index % 5);
    kernel = Mat::ones(Size(ksize, ksize), CV_32F) / (float)(ksize*ksize);
    filter2D(src, dest, -1, kernel, Point(-1, -1), 0);
    imshow(OUTPUT_WIN, dest);
    index++;
}
return 0;
```

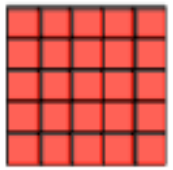
为梦想增值！

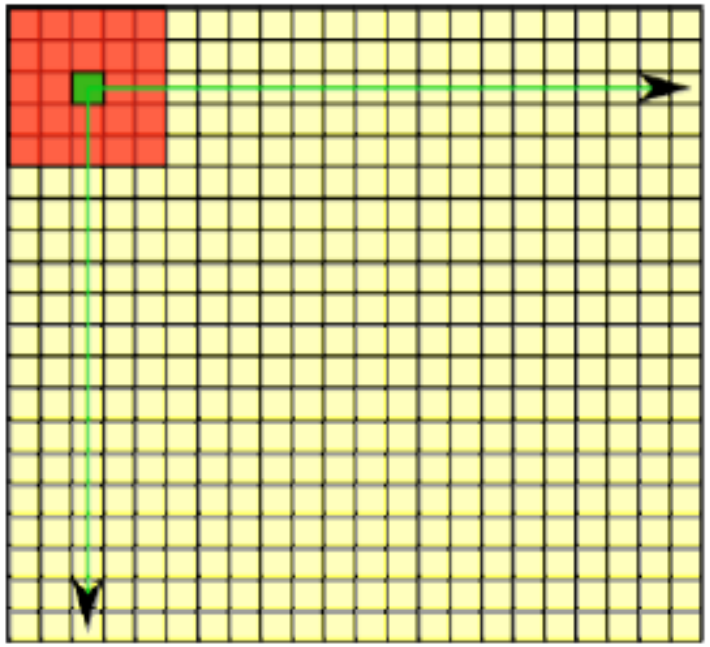
16.处理边缘

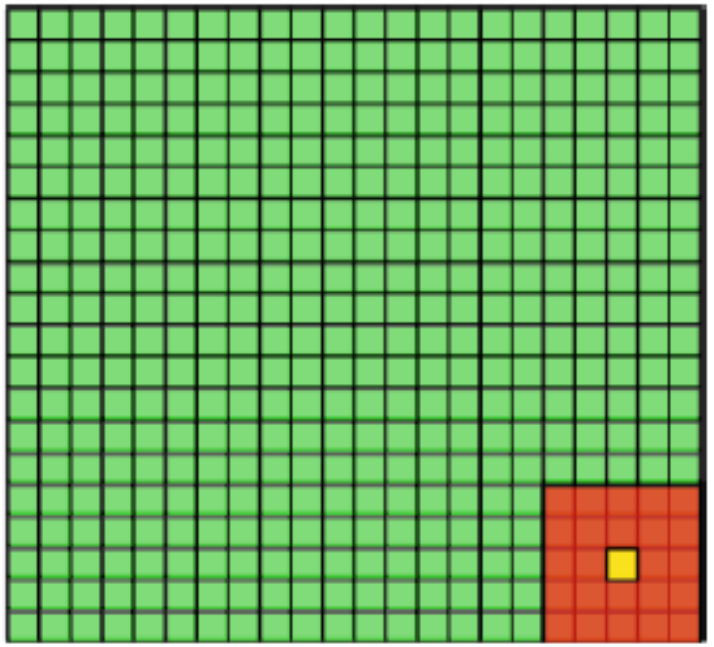
- 卷积边缘问题
- 处理边缘
- 代码演示

为梦想增值！

卷积边缘问题







为梦想增值！

卷积边界问题

- 图像卷积的时候边界像素，不能被卷积操作，原因在于边界像素没有完全跟kernel重叠，所以当3x3滤波时候有1个像素的边缘没有被处理，5x5滤波的时候有2个像素的边缘没有被处理。

处理边缘

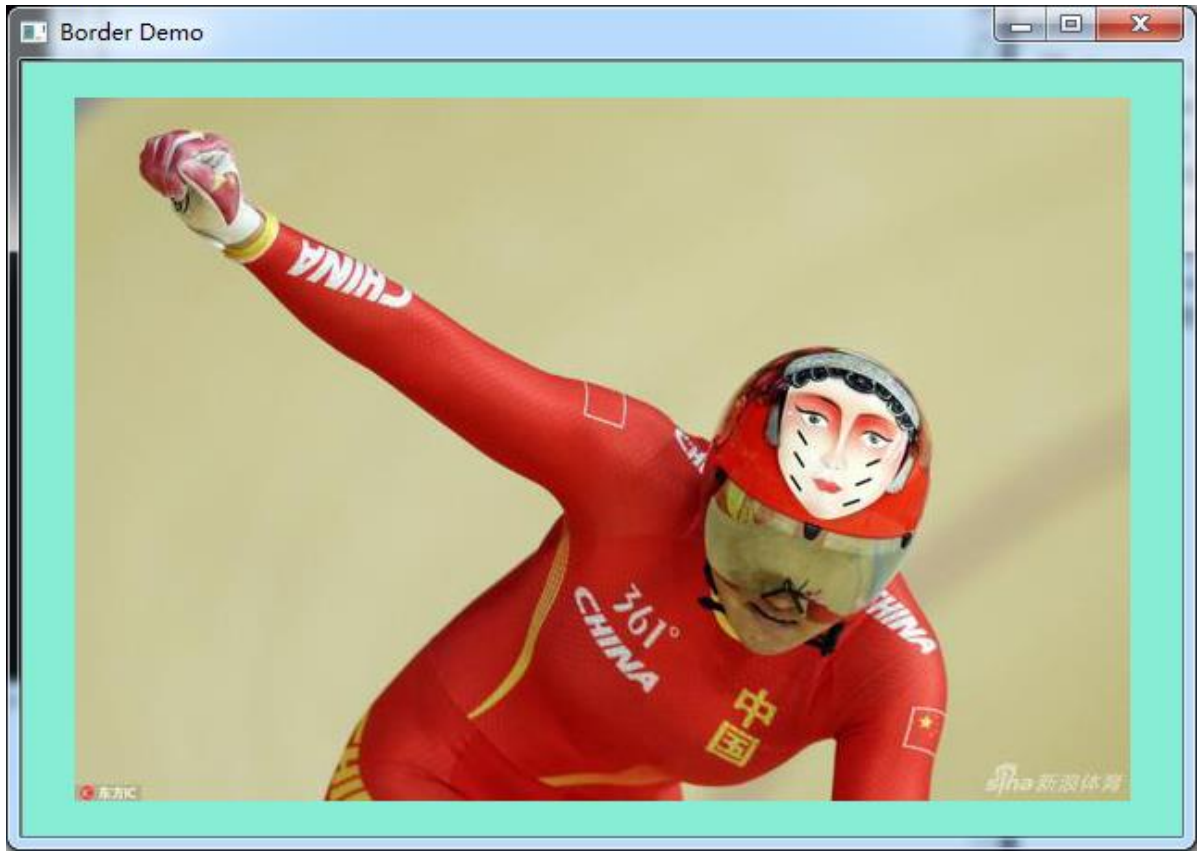
在卷积开始之前增加边缘像素，填充的像素值为0或者RGB黑色，比如3x3在四周各填充1个像素的边缘，这样就确保图像的边缘被处理，在卷积处理之后再去掉这些边缘。openCV中默认的处理方法是： BORDER_DEFAULT，此外常用的还有如下几种:

- **BORDER_CONSTANT** – 填充边缘用指定像素值
- **BORDER_REPLICATE –** 填充边缘像素用已知的边缘像素值。
- **BORDER_WRAP** – 用另外一边的像素来补偿填充

BORDER_DEFAULT

为梦想增值！

BORDER_CONSTANT



为梦想增值！

BORDER_REPLICATE – 通过插值计算

为梦想增值！

BORDER_WRAP – 另外一边补偿

为梦想增值！

API说明 – 给图像添加边缘API

- copyMakeBorder（
  - Mat src, // 输入图像
  - Mat dst, // 添加边缘图像
  - int top, // 边缘长度，一般上下左右都取相同值，
  - int bottom,
  - int left,
  - int right,
  - int borderType // 边缘类型
  - Scalar value
)

演示代码

```
int flag = 0;
// int border_type = BORDER_DEFAULT;
int border_type = BORDER_WRAP;
RNG rng;
Scalar color;
int top = (int)(0.05*src.rows);
int bottom = (int)(0.05*src.rows);
int left = (int)(0.05*src.cols);
int right = (int)(0.05*src.cols);

while (true) {
    flag = waitKey(500);
    if ((char)flag == 27) {
        break;
    }
    if ((char)flag == 'c') {
        border_type = BORDER_CONSTANT;
    }
    else if ((char)flag == 'r') {
        border_type = BORDER_REPLICATE;
    }
    color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));
    copyMakeBorder(src, dest, top, bottom, left, right, border_type, color);
    imshow(OUTPUT_WIN, dest);
}
return 0;
```

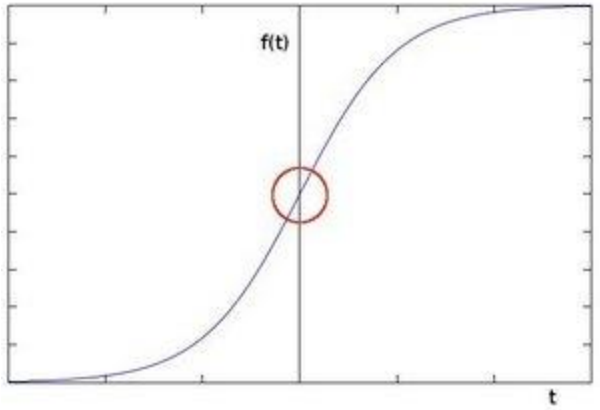
为梦想增值！

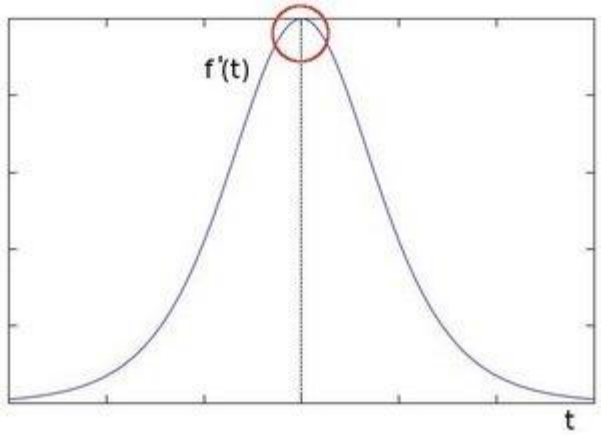
17.Sobel算子

- 卷积应用-图像边缘提取
- 相关API
- 代码演示

为梦想增值！

卷积应用-图像边缘提取





为梦想增值！

卷积应用-图像边缘提取

- 边缘是什么 – 是像素值发生跃迁的地方，是图像的显著特征之一，在图像特征提取、对象检测、模式识别等方面都有重要的作用。
- 如何捕捉/提取边缘 – 对图像求它的一阶导数

  delta = f(x) – f(x-1), delta越大，说明像素在X方向变化越大，边缘信号越强，
- 我已经忘记啦，不要担心，用Sobel算子就好！卷积操作！

Sobel算子

- 是离散微分算子（discrete differentiation operator），用来计算图像灰度的近似梯度
- Soble算子功能集合高斯平滑和微分求导
- 又被称为一阶微分算子，求导算子，在水平和垂直两个方向上求导，得到图像X方法与Y方向梯度图像

Sobel算子

水平梯度

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

垂直梯度

$$G = \sqrt{G_x^2 + G_y^2}$$

最终图像梯度

$$G = |G_x| + |G_y|$$

为梦想增值！

Sobel算子

- 求取导数的近似值，kernel=3时不是很准确，OpenCV使用改进版本Scharr函数，算子如下：

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

API说明cv::Sobel

```
cv::Sobel (
InputArray Src // 输入图像
OutputArray dst// 输出图像，大小与输入图像一致
int depth // 输出图像深度.
Int dx.  // X方向，几阶导数
int dy // Y方向，几阶导数.
int ksize, SOBEL算子kernel大小，必须是1、3、5、7
double scale  = 1
double delta = 0
int borderType = BORDER_DEFAULT
)
```

Input depth (src.depth())	Output depth (ddepth)
CV_8U	-1/CV_16S/CV_32F/CV_64F
CV_16U/CV_16S	-1/CV_32F/CV_64F
CV_32F	-1/CV_32F/CV_64F
CV_64F	-1/CV_64F

为梦想增值！

API说明cv::Scharr

```
cv::Scharr (
InputArray Src // 输入图像
OutputArray dst// 输出图像，大小与输入图像一致
int depth // 输出图像深度.
Int dx. // X方向，几阶导数
int dy // Y方向，几阶导数.
double scale  = 1
double delta = 0
int borderType = BORDER_DEFAULT
)
```

其它API

- GaussianBlur(src, dst, Size(3,3), 0, 0, BORDER_DEFAULT);
- cvtColor(src, gray, COLOR_RGB2GRAY);
- addWeighted(A, 0.5,B, 0.5, 0, AB);
- convertScaleAbs(A, B)// 计算图像A的像素绝对值，输出到图像B

$$\texttt{dst}(I) = \texttt{saturate\_cast<uchar>}(|\texttt{src}(I) * \texttt{alpha} + \texttt{beta}|)$$

为梦想增值！

演示代码

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

    char INPUT_TITLE[] = "input image";
    char OUTPUT_TITLE[] = "sobel-demo";
    namedWindow(INPUT_TITLE, CV_WINDOW_AUTOSIZE);
    namedWindow(OUTPUT_TITLE, CV_WINDOW_AUTOSIZE);

    GaussianBlur(src, src, Size(3, 3), 0, 0, BORDER_DEFAULT);
    cvtColor(src, src, CV_BGR2GRAY);
    imshow(INPUT_TITLE, src);

    Mat xgrad, ygrad;
    Sobel(src, xgrad, CV_16S, 1, 0, 3);
    Sobel(src, ygrad, CV_16S, 0, 1, 3);

    convertScaleAbs(xgrad, xgrad);
    convertScaleAbs(ygrad, ygrad);

    addWeighted(xgrad, 0.5, ygrad, 0.5, 0, dest);
    imshow(OUTPUT_TITLE, dest);

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

为梦想增值！

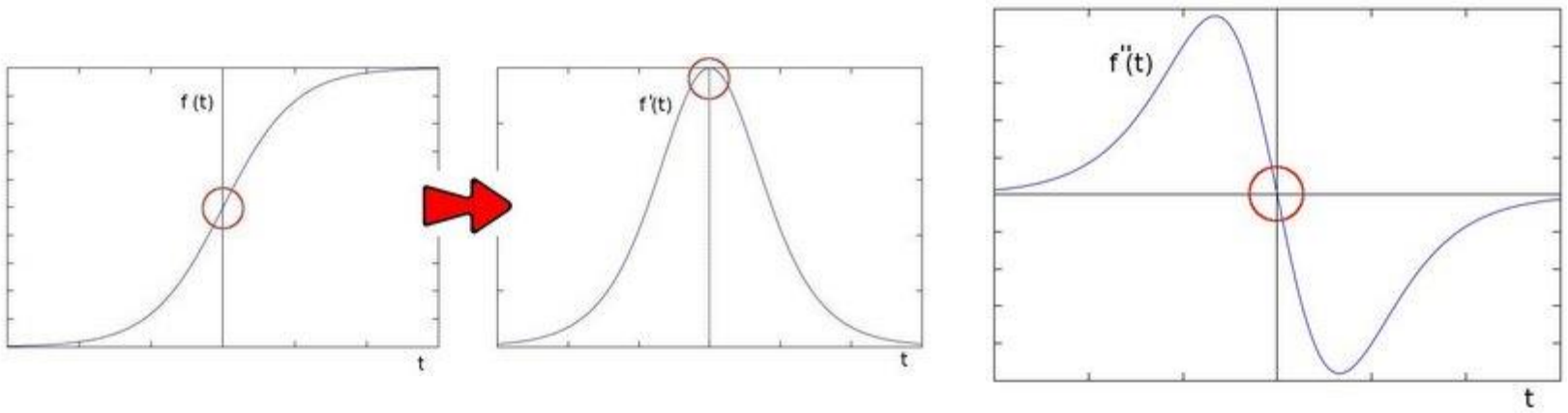
为梦想增值！

18.Laplance算子

- 理论
- API使用
- 代码演示

为梦想增值！

理论



解释：在二阶导数的时候，最大变化处的值为零即边缘是零值。通过二阶导数计算，依据此理论我们可以计算图像二阶导数，提取边缘。

为梦想增值！

Laplance算子

- 二阶导数我不会，别担心 ->拉普拉斯算子(Laplance operator)

$$Laplace(f) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- Opencv已经提供了相关API - cv::Laplance

为梦想增值！

处理流程

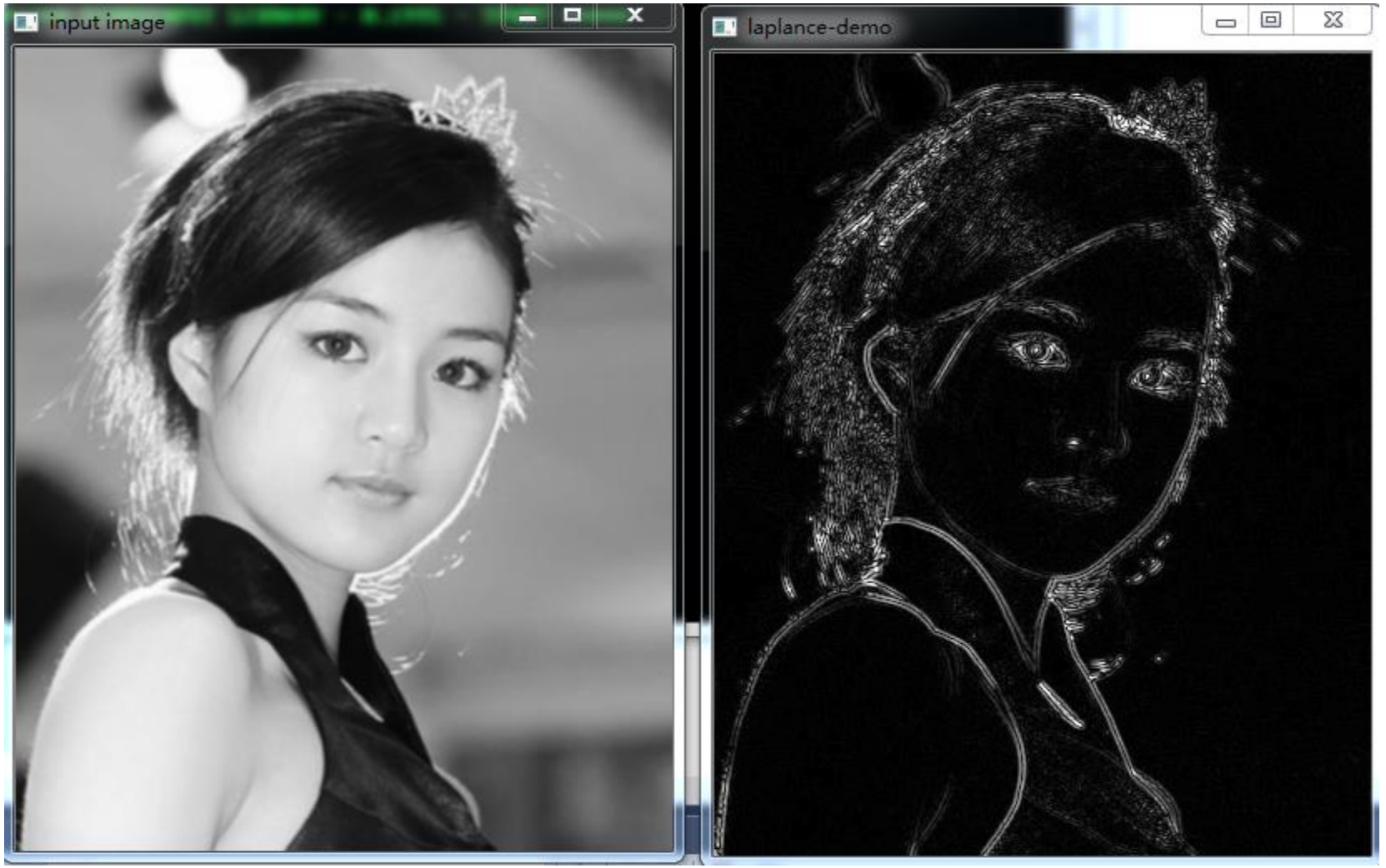
- 高斯模糊 – 去噪声GaussianBlur()
- 转换为灰度图像cvtColor()
- 拉普拉斯 – 二阶导数计算Laplacian()
- 取绝对值convertScaleAbs()
- 显示结果

API使用cv::Laplacian

```
Laplacian(
InputArray src,
OutputArray dst,
int depth, //深度CV_16S
int kisze, // 3
double scale = 1,
double delta =0.0,
int borderType = 4
)
```

为梦想增值！

显示图像



为梦想增值！

演示代码

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

char INPUT_TITLE[] = "input image";
char OUTPUT_TITLE[] = "laplance-demo";
namedWindow(INPUT_TITLE, CV_WINDOW_AUTOSIZE);
namedWindow(OUTPUT_TITLE, CV_WINDOW_AUTOSIZE);

// pre-process, blur
GaussianBlur(src, src, Size(3, 3), 0);

// convert to gray
cvtColor(src, src, CV_BGR2GRAY);

// display input
imshow(INPUT_TITLE, src);

// laplance
Laplacian(src, dest, CV_16S, 3);
convertScaleAbs(dest, dest);

imshow(OUTPUT_TITLE, dest);

waitKey(0);
return 0;
```

为梦想增值！

19.Canny边缘检测

- Canny算法介绍
- API cv::Canny()
- 代码演示

为梦想增值！

Canny算法介绍

- **Canny**是边缘检测算法，在1986年提出的。
- 是一个很好的边缘检测器
- 很常用也很实用的图像处理方法

Canny算法介绍 – 五步 in cv::Canny

1. 高斯模糊 - GaussianBlur
2. 灰度转换 - cvtColor
3. 计算梯度 – Sobel/Scharr
4. 非最大信号抑制
5. 高低阈值输出二值图像

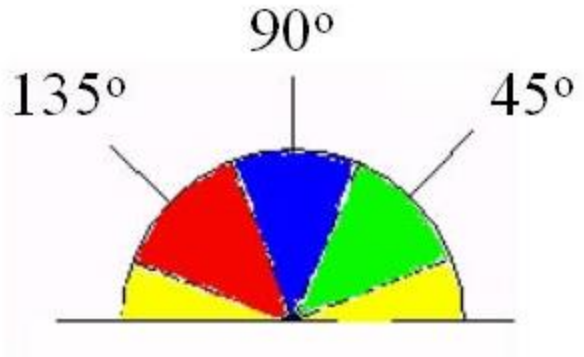
Canny算法介绍 - 非最大信号抑制

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

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

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \arctan(\frac{G_y}{G_x})$$



其中黄色区域取值范围为0~22.5 与157.5~180

绿色区域取值范围为22.5 ~ 67.5

蓝色区域取值范围为67.5~112.5

红色区域取值范围为112.5~157.5

为梦想增值！

Canny算法介绍-高低阈值输出二值图像

- T1， T2为阈值，凡是高于T2的都保留，凡是小于T1都丢弃，从高于T2的像素出发，凡是大于T1而且相互连接的，都保留。最终得到一个输出二值图像。
- 推荐的高低阈值比值为 T2: T1 = 3:1/2:1其中T2为高阈值，T1为低阈值

API – cv::Canny

Canny（
InputArray src, // 8-bit的输入图像
OutputArray edges,// 输出边缘图像， 一般都是二值图像，背景是黑色
double threshold1,// 低阈值，常取高阈值的1/2或者1/3
double threshold2,// 高阈值
int aptertureSize,// Soble算子的size，通常3x3，取值3
bool L2gradient // 选择 true表示是L2来归一化，否则用L1归一化
）

$$L_2 \text{ norm} = \sqrt{(dI/dx)^2 + (dI/dy)^2}$$

$$L_1 \text{ norm} = |dI/dx| + |dI/dy|$$

默认情况一般选择是L1，参数设置为false

演示代码

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

    char INPUT_TITLE[] = "input image";
    namedWindow(INPUT_TITLE, CV_WINDOW_AUTOSIZE);
    namedWindow(OUTPUT_TITLE, CV_WINDOW_AUTOSIZE);
    imshow(INPUT_TITLE, src);

    dest.create(src.size(), src.type());
    cvtColor(src, gray_src, CV_BGR2GRAY);
    createTrackbar("Threshold :", OUTPUT_TITLE, &high_threshold, max_value, CannyDemo);
    CannyDemo(0, 0);

    waitKey(0);
}

void CannyDemo(int, void*) {
    blur(gray_src, edges_dest, Size(3, 3), Point(-1, -1), BORDER_DEFAULT);
    double lowt = high_threshold / rate;
    Canny(edges_dest, edges_dest, lowt, high_threshold, 3, true);
    dest = Scalar::all(0);

    // 使用遮罩层，只有非零的元素才被copy到模板中。
    src.copyTo(dest, edges_dest);
    imshow(OUTPUT_TITLE, dest);
}
```

为梦想增值！

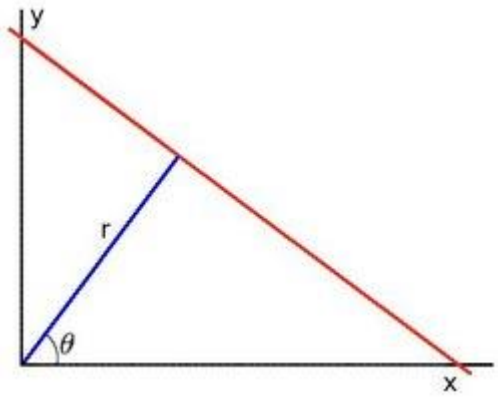
20.霍夫变换-直线

- 霍夫直线变换介绍
- 相关API学习
- 代码演示

为梦想增值！

霍夫直线变换介绍

- **Hough Line Transform**用来做直线检测
- 前提条件 – 边缘检测已经完成
- 平面空间到极坐标空间转换



$x=\rho\cos\theta, y=\rho\sin\theta$

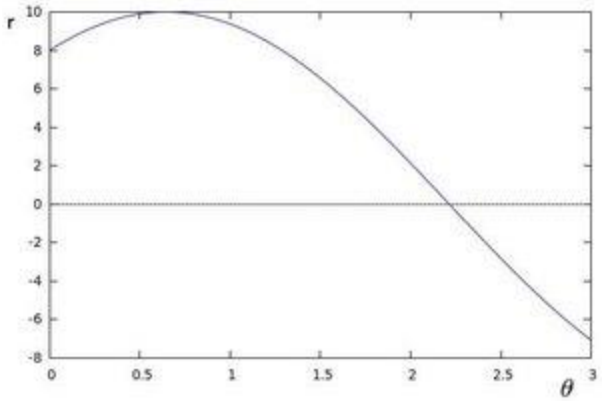
$\rho^2=x^2+y^2,\ \tan\theta=y/x\ (x\neq0)$

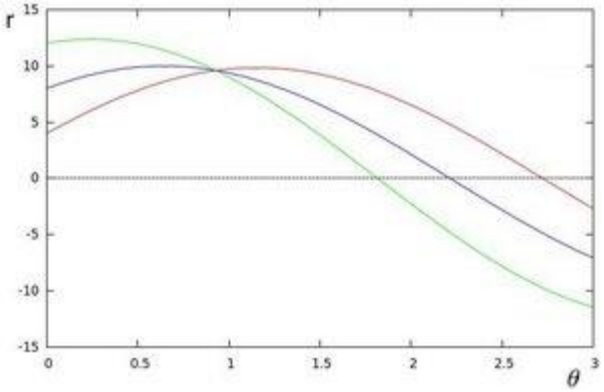
霍夫直线变换介绍

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

$$r = x\cos\theta + y\sin\theta$$

$$r_\theta = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$$



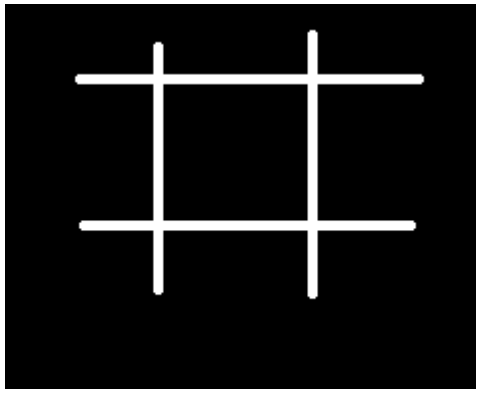


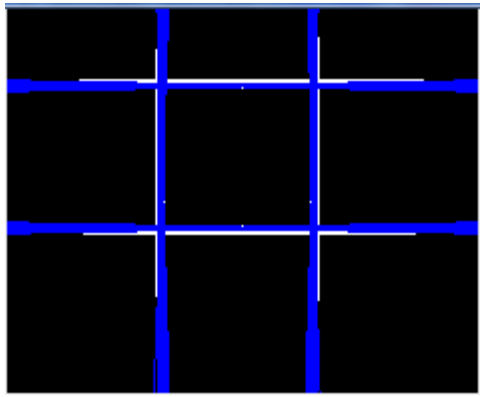
为梦想增值！

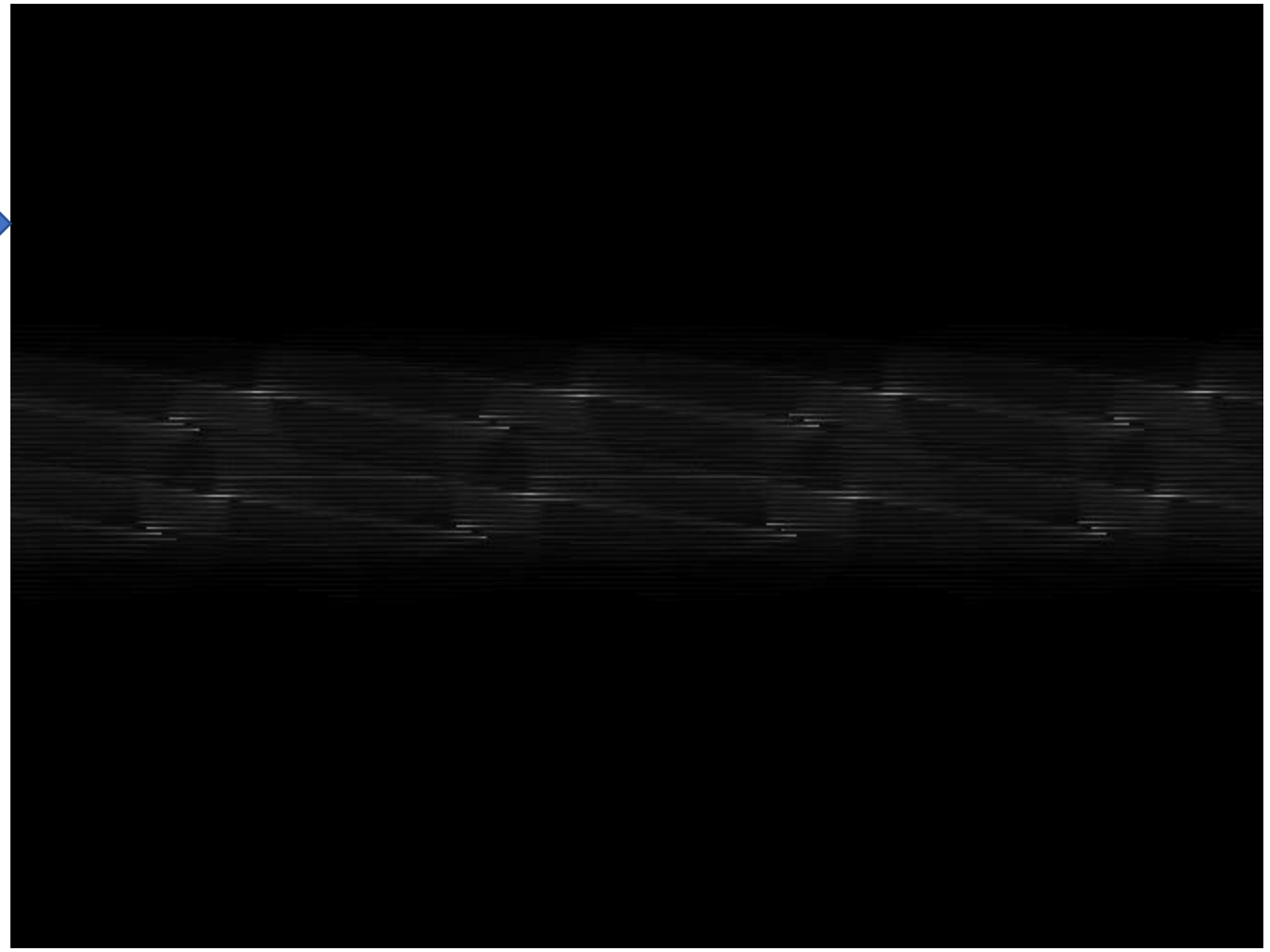
霍夫直线变换介绍

- 对于任意一条直线上的所有点来说
- 变换到极坐标中，从[0~360]空间，可以得到r的大小
- 属于同一条直线上点在极坐标空(r, theta)必然在一个点上有最强的信号出现，根据此反算到平面坐标中就可以得到直线上各点的像素坐标。从而得到直线

从平面坐标变换到霍夫空间（极坐标）







为梦想增值！

相关API学习

- 标准的霍夫变换 cv::HoughLines从平面坐标转换到霍夫空间，最终输出是(θ, r_θ) 表示极坐标空间
- 霍夫变换直线概率 cv::HoughLinesP最终输出是直线的两个点 (x_0, y_0, x_1, y_1)

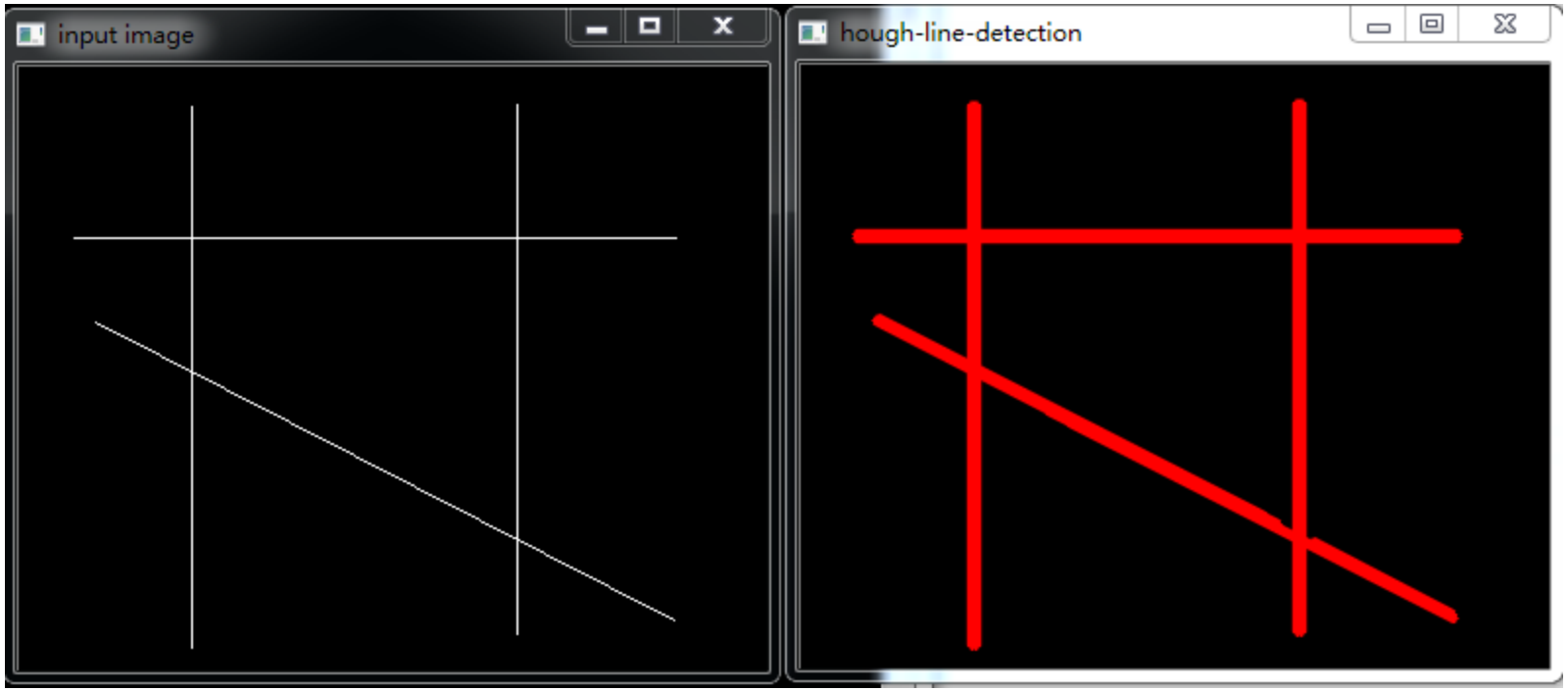
相关API学习

```
cv::HoughLines(
InputArray src, // 输入图像，必须8-bit的灰度图像
OutputArray lines, // 输出的极坐标来表示直线
double rho, // 生成极坐标时候的像素扫描步长
double theta, //生成极坐标时候的角度步长，一般取值CV_PI/180
int threshold, // 阈值，只有获得足够交点的极坐标点才被看成是直线
double srn=0;// 是否应用多尺度的霍夫变换，如果不是设置0表示经典霍夫变换
double stn=0;//是否应用多尺度的霍夫变换，如果不是设置0表示经典霍夫变换
double min_theta=0; // 表示角度扫描范围 0 ~180之间， 默认即可
double max_theta=CV_PI
) // 一般情况是有经验的开发者使用，需要自己反变换到平面空间
```

相关API学习

```
cv::HoughLinesP(
InputArray src, // 输入图像，必须8-bit的灰度图像
OutputArray lines, // 输出的极坐标来表示直线
double rho, // 生成极坐标时候的像素扫描步长
double theta, //生成极坐标时候的角度步长，一般取值CV_PI/180
int threshold, // 阈值，只有获得足够交点的极坐标点才被看成是直线
double minLineLength=0;// 最小直线长度
double maxLineGap=0;// 最大间隔
)
```

HoughLinesP检测效果



为梦想增值！

演示代码

先进行边缘检测

```
imshow(INPUT_TITLE, src);
Canny(src, src_gray, 150, 200, 3);
cvtColor(src_gray, dest, CV_GRAY2BGR);
```

霍夫直线检测

```
//
vector<Vec4f> plines;
HoughLinesP(src_gray, plines, 1, CV_PI / 180, 10, 0, 10);
for (size_t i = 0; i < plines.size(); i++) {
    Vec4f hl = plines[i];
    line(dest, Point(hl[0], hl[1]), Point(hl[2], hl[3]), Scalar(0, 0, 255), 3, CV_AA);
}
imshow(OUTPUT_TITLE, dest);
```

为梦想增值！

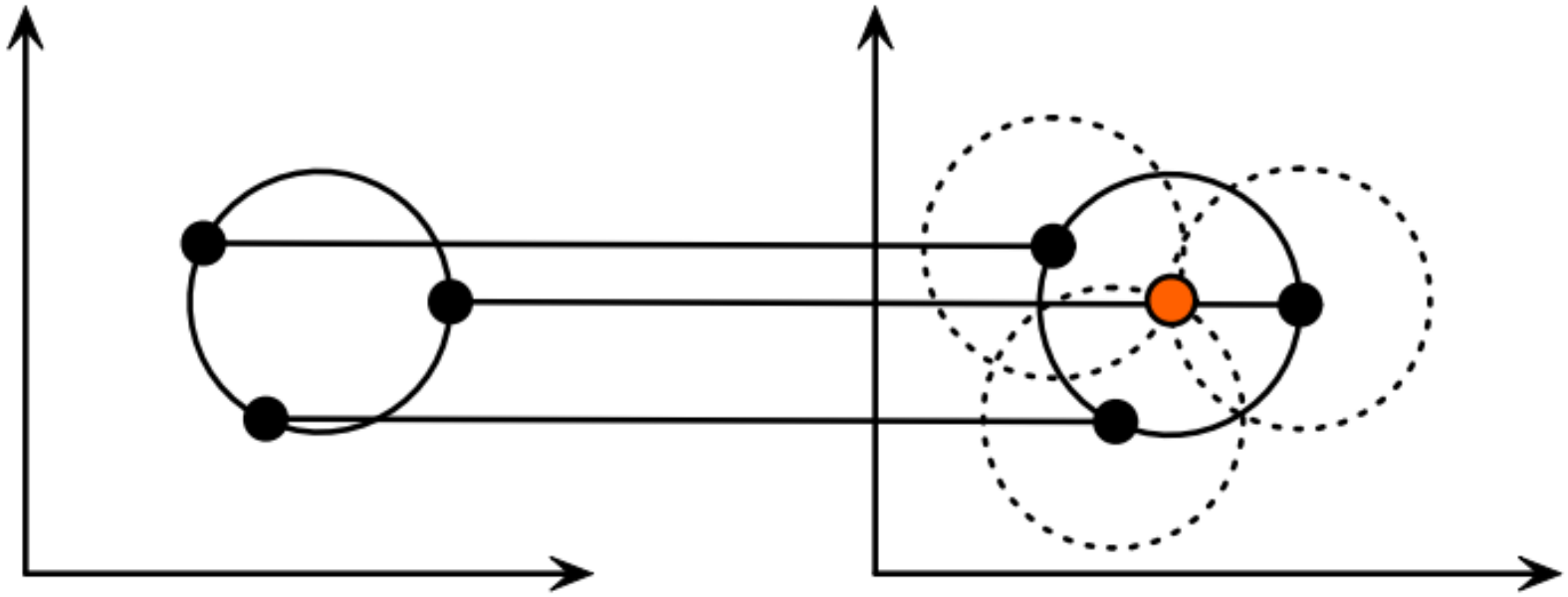
21.霍夫圆变换

- 霍夫圆检测原理
- 相关API
- 代码演示

为梦想增值！

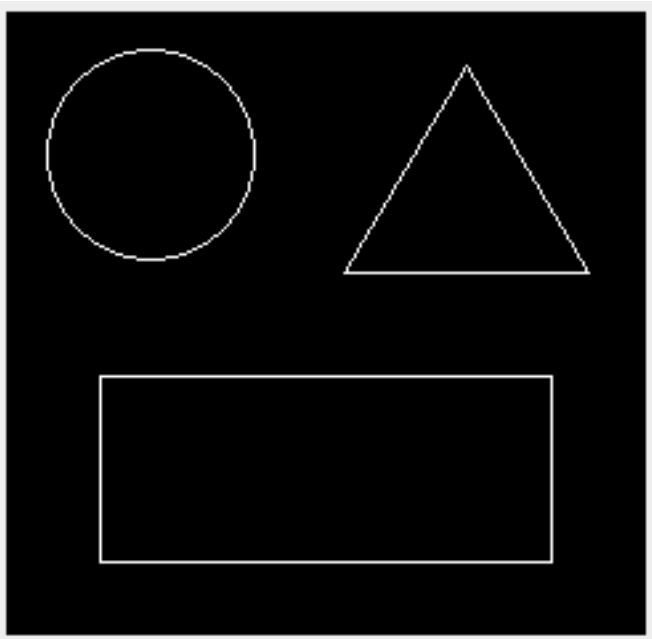
霍夫圆检测原理

$$x = a + R\cos(\theta)$$
$$y = b + R\sin(\theta)$$

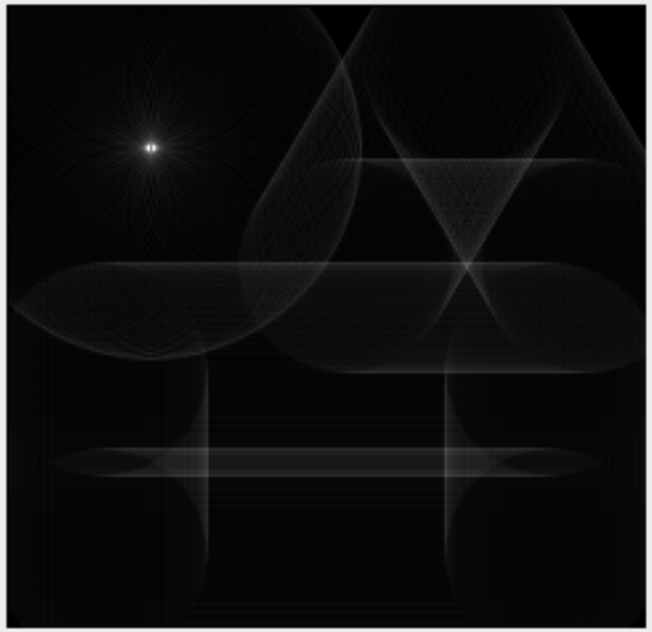


Hough[theta][a][b]++

为梦想增值！



原图



极坐标空间效果,最亮一点表示圆心

为梦想增值!

霍夫圆变换原理

- **从平面坐标到极坐标转换三个参数** $C(x_0, y_0, r)$其中x_0, y_0是圆心
- **假设平面坐标的任意一个圆上的点，转换到极坐标中：**

  $C(x_0, y_0, r)$ 处有最大值，霍夫变换正是利用这个原理实现圆的检测。

相关API cv::HoughCircles

- 因为霍夫圆检测对噪声比较敏感，所以首先要对图像做中值滤波。
- 基于效率考虑，Opencv中实现的霍夫变换圆检测是基于图像梯度的实现，分为两步：

  1. 检测边缘，发现可能的圆心

  2. 基于第一步的基础上从候选圆心开始计算最佳半径大小

HoughCircles参数说明

HoughCircles(
InputArray image, // 输入图像 ,必须是8位的单通道灰度图像
OutputArray circles, // 输出结果，发现的圆信息
Int method, // 方法 - HOUGH_GRADIENT
Double dp, // dp = 1;
Double mindist, // 10 最短距离-可以分辨是两个圆的，否则认为是同心圆- *src_gray.rows/8*
Double param1, // canny edge detection low threshold
Double param2, // 中心点累加器阈值 – 候选圆心
Int minradius, // 最小半径
Int maxradius//最大半径
)

为梦想增值！

演示代码

```
char INPUT_TITLE[] = "input image";
char OUTPUT_TITLE[] = "hough circle demo";
namedWindow(INPUT_TITLE, CV_WINDOW_AUTOSIZE);
namedWindow(OUTPUT_TITLE, CV_WINDOW_AUTOSIZE);

// show input image
// medianBlur(src, src, 5);
cvtColor(src, src, CV_BGR2GRAY);
GaussianBlur(src, dest, Size(5, 5), 0, 0);
imshow(INPUT_TITLE, src);

// 基于灰度空间
vector<Vec3f> circles;
HoughCircles(dest, circles, HOUGH_GRADIENT, 1, 10, 100, 30, 5, 50);
// 重新转回到RGB色彩空间
cvtColor(dest, dest, CV_GRAY2BGR);
for (size_t i = 0; i < circles.size(); i++) {
    Vec3f c3 = circles[i];
    circle(dest, Point(c3[0], c3[1]), c3[2], Scalar(0, 0, 255), 3, LINE_AA);
    circle(dest, Point(c3[0], c3[1]), 2, Scalar(0, 0, 255), 3, LINE_AA);
}
imshow(OUTPUT_TITLE, dest);

waitKey(0);
return 0;
```

为梦想增值！

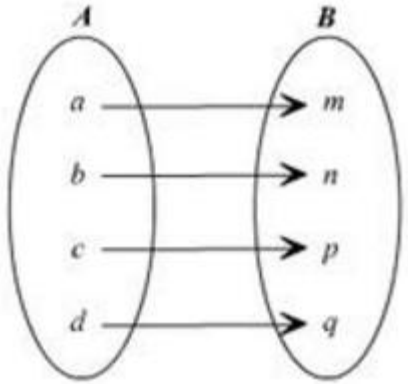
22.像素重映射(cv::remap)

- 什么是像素重映射
- API介绍
- 代码演示

为梦想增值！

什么是像素重映射

- **简单点说就是把输入图像中各个像素按照一定的规则映射到另外一张图像的对应位置上去，形成一张新的图像。**



$$g(x,y) = f(h(x,y))$$

g(x,y)是重映射之后的图像，h(x,y)是功能函数，f是源图像

为梦想增值！

什么是像素重映射

假设有映射函数 $h(x,y) = (I.cols - x, y)$

为梦想增值！

API介绍cv::remap

```
Remap(
InputArray src,// 输入图像
OutputArray dst,// 输出图像
InputArray  map1,// x 映射表 CV_32FC1/CV_32FC2
InputArray map2,// y 映射表
int interpolation,// 选择的插值方法，常见线性插值，可选择立方等
int borderMode,// BORDER_CONSTANT
const Scalar borderValue// color
)
```

为梦想增值！

API介绍cv::remap

```
if (x > src.cols*0.25 && x < src.cols*0.75 && y > src.rows*0.25 && y < src.rows*0.75) {
    map_x.at<float>(y, x) = 2 * (x - src.cols*0.25f) + 0.5f;
    map_y.at<float>(y, x) = 2 * (y - src.rows*0.25f) + 0.5f;
} else {
    map_x.at<float>(y, x) = 0;
    map_y.at<float>(y, x) = 0;
}
```

缩小一半

```
map_x.at<float>(y, x) = (float)x;
map_y.at<float>(y, x) = (float)(src.rows - y);
```

Y方向对调

```
map_x.at<float>(y, x) = (float)(src.cols - x);
map_y.at<float>(y, x) = (float)y;
```

X方向对调

```
map_x.at<float>(y, x) = (float)(src.cols - x);
map_y.at<float>(y, x) = (float)(src.rows - y);
```

XY方向同时对调

为梦想增值！

演示代码

```
int main(int argc, char** argv) {
    src = imread("D:/vcprojects/images/test.png");
    if (!src.data) {
        printf("could not load image...\n");
        return -1;
    }
    char input_win[] = "input image";
    namedWindow(input_win, CV_WINDOW_AUTOSIZE);
    namedWindow(OUTPUT_TITLE, CV_WINDOW_AUTOSIZE);
    imshow(input_win, src);
    map_x.create(src.size(), CV_32FC1);
    map_y.create(src.size(), CV_32FC1);
    while (true) {
        index = waitKey(500);
        if ((char)index == 27) {
            break;
        }
        update_remap();
        remap(src, dst, map_x, map_y, INTER_LINEAR, BORDER_CONSTANT, Scalar(0, 255, 255));
        imshow(OUTPUT_TITLE, dst);
    }

    return 0;
}
```

为梦想增值！

23.直方图均衡化

- 什么是直方图
- 直方图均衡化
- API说明
- 代码演示

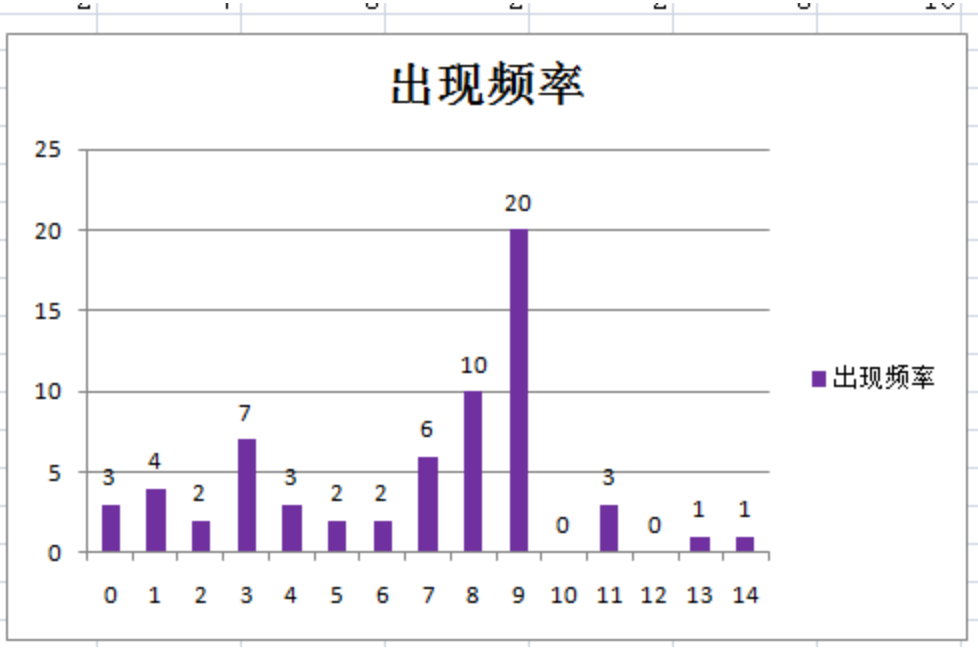
为梦想增值！

什么是直方图(Histogram)

1	2	3	5	6	7	9	4
2	3	4	1	0	0	0	9
3	3	3	9	1	11	3	3
8	8	8	9	11	13	8	8
8	8	8	9	1	6	8	8
7	7	7	9	4	5	7	7
9	9	9	9	14	9	9	9
9	9	9	9	9	11	9	9

假设有图像数据8x8，像素值范围0~14共15个灰度等级，统计得到各个等级出现次数及直方图如右侧所示

像素等级	出现频率
0	3
1	4
2	2
3	7
4	3
5	2
6	2
7	6
8	10
9	20
10	0
11	3
12	0
13	1
14	1

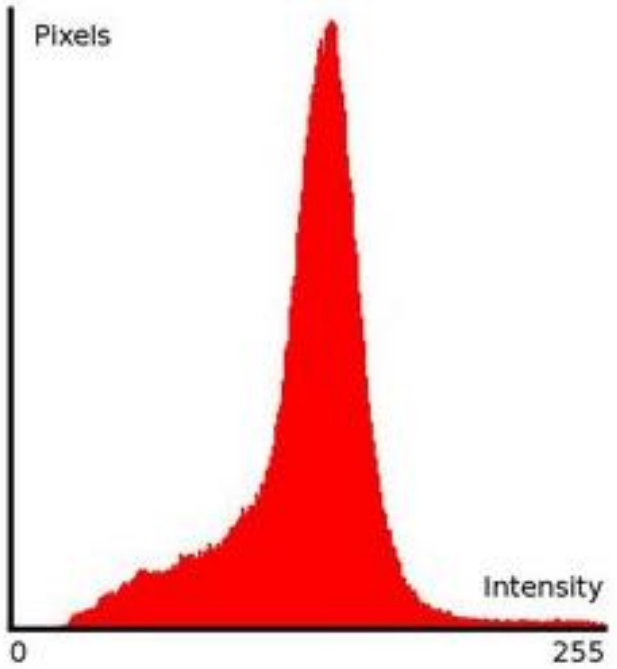


为梦想增值！

什么是直方图

- 图像直方图，是指对整个图像像在灰度范围内的像素值(0~255)统计出现频率次数，据此生成的直方图，称为图像直方图-直方图。直方图反映了图像灰度的分布情况。是图像的统计学特征。

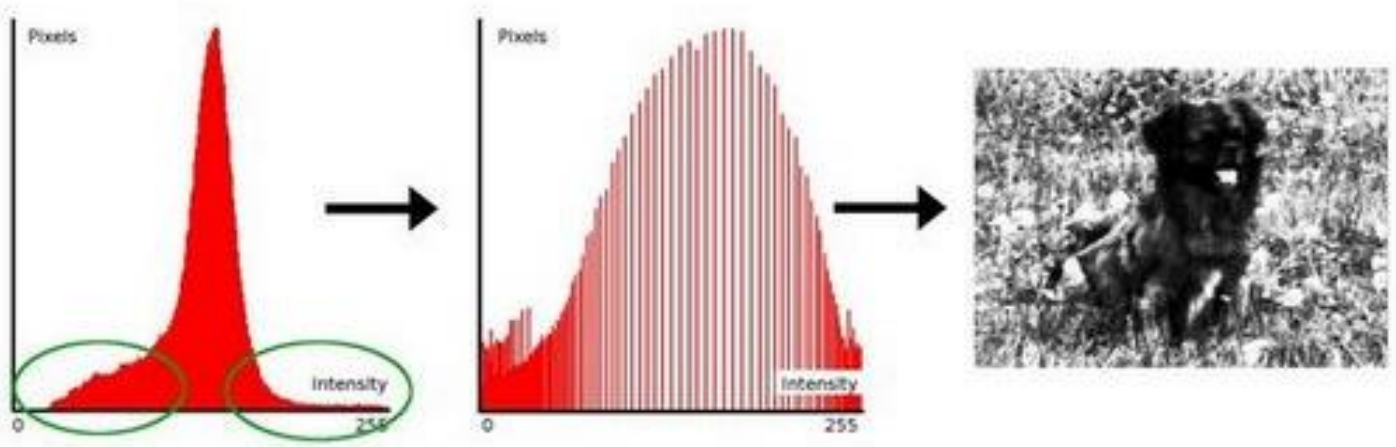
图像直方图



为梦想增值！

直方图均衡化

是一种提高图像对比度的方法，拉伸图像灰度值范围。



为梦想增值！

直方图均衡化

- 如何实现，通过上一课中的remap我们知道可以将图像灰度分布从一个分布映射到另外一个分布，然后在得到映射后的像素值即可。

$$H^{'}(i) = \sum_{0 \le j < i} H(j)$$

$$equalized(x,y) = H^{'}(src(x,y))$$

API说明cv::equalizeHist

```
equalizeHist(
InputArray src,//输入图像，必须是8-bit的单通道图像
OutputArray dst// 输出结果
)
```

为梦想增值！

演示代码

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

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

    cvtColor(src, src, CV_BGR2GRAY);
    equalizeHist(src, dst);
    char INPUT_T[] = "input image";
    char OUTPUT_T[] = "result image";
    namedWindow(INPUT_T, CV_WINDOW_AUTOSIZE);
    namedWindow(OUTPUT_T, CV_WINDOW_AUTOSIZE);

    imshow(INPUT_T, src);
    imshow(OUTPUT_T, dst);

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

为梦想增值！

直方图均衡化

- 什么是直方图
- 直方图均衡化
- API说明
- 代码演示

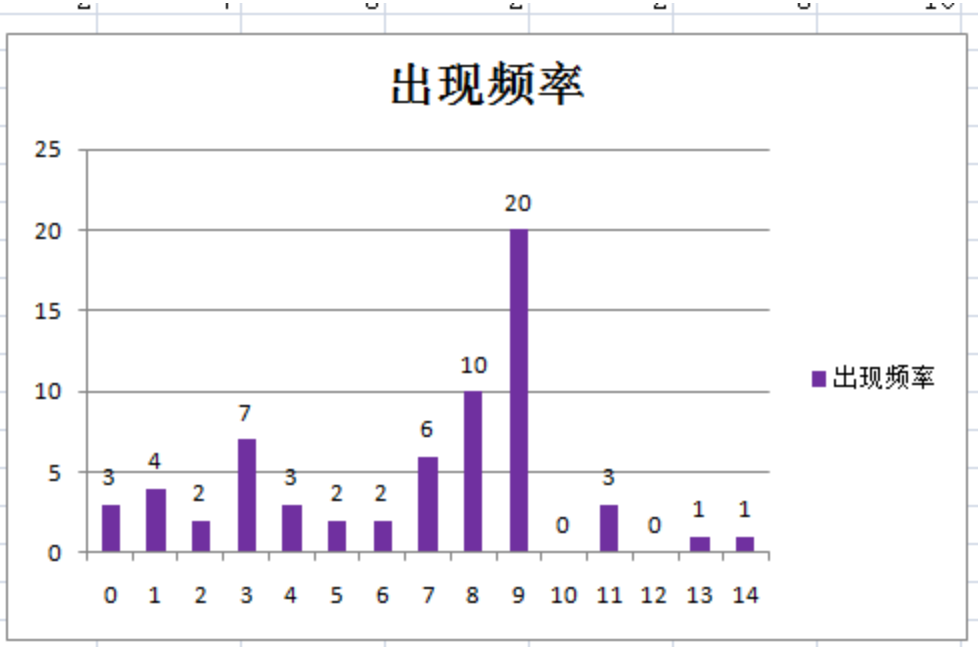
为梦想增值！

什么是直方图(Histogram)

1	2	3	5	6	7	9	4
2	3	4	1	0	0	0	9
3	3	3	9	1	11	3	3
8	8	8	9	11	13	8	8
8	8	8	9	1	6	8	8
7	7	7	9	4	5	7	7
9	9	9	9	14	9	9	9
9	9	9	9	9	11	9	9

假设有图像数据8x8，像素值范围0~14共15个灰度等级，统计得到各个等级出现次数及直方图如右侧所示

像素等级	出现频率
0	3
1	4
2	2
3	7
4	3
5	2
6	2
7	6
8	10
9	20
10	0
11	3
12	0
13	1
14	1

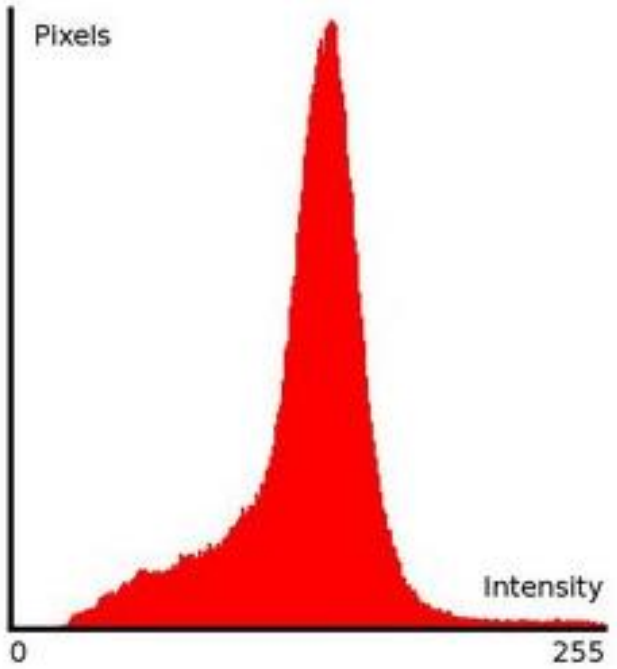


为梦想增值！

什么是直方图

- 图像直方图，是指对整个图像像在灰度范围内的像素值(0~255)统计出现频率次数，据此生成的直方图，称为图像直方图-直方图。直方图反映了图像灰度的分布情况。是图像的统计学特征。

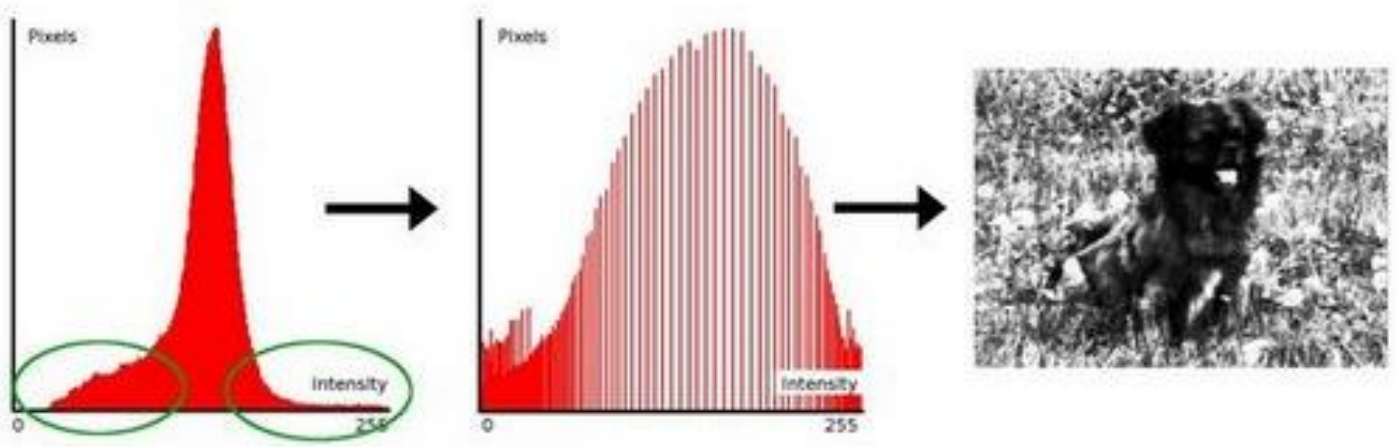
图像直方图



为梦想增值！

直方图均衡化

是一种提高图像对比度的方法，拉伸图像灰度值范围。



为梦想增值！

直方图均衡化

- 如何实现，通过上一课中的remap我们知道可以将图像灰度分布从一个分布映射到另外一个分布，然后在得到映射后的像素值即可。

$$H^{'}(i) = \sum_{0 \le j < i} H(j)$$

$$equalized(x,y) = H^{'}(src(x,y))$$

API说明cv::equalizeHist

```
equalizeHist(
InputArray src,//输入图像，必须是8-bit的单通道图像
OutputArray dst// 输出结果
)
```

为梦想增值！

演示代码

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

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

    cvtColor(src, src, CV_BGR2GRAY);
    equalizeHist(src, dst);
    char INPUT_T[] = "input image";
    char OUTPUT_T[] = "result image";
    namedWindow(INPUT_T, CV_WINDOW_AUTOSIZE);
    namedWindow(OUTPUT_T, CV_WINDOW_AUTOSIZE);

    imshow(INPUT_T, src);
    imshow(OUTPUT_T, dst);

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

为梦想增值！