湖南科技大学课程教案
（章节、专题首页）

授课教师：王志喜　　职称：副教授　　单位：计算机科学与工程学院

课 程 名 称	计算机图形图像技术
章节、专题	图像解析
教学目标及基 本 要 求	掌握最常用的图像特征统计工具和二值化、边缘检测、轮廓检测等常用的图像分割方法。
教 学 重 点	图像的灰度直方图，图像的二值化
教 学 难 点	图像的灰度直方图，图像的二值化，轮廓检测
教学内容与时 间 分 配	（1）图像的特征统计（1课时） （2）图像的二值化（0.6课时） （3）图像的特征提取（0.6课时） （4）图像的特征描述（0.8课时） （5）图像中的景物匹配（1课时） 共计4课时。
习　　题	第16.7.1节（基础知识题）和16.7.2节（程序设计题）。

第16章　图像解析

16.1　图像解析的主要内容

图像解析的目的是从图像数据中找到某些有用的东西，它的内容通常包括特征提取、特征描述、景物匹配和对象识别等几个部分。

- 特征提取的主要任务是将图像中有意义的特征部分（如图像中的边缘和区域等）提取出来。
- 特征描述的主要任务是用数据、符号、形式语言来表示从图像中提取出来的一些具有不同特征的小区域。
- 景物匹配的主要任务是在源图像中寻找与指定景物最匹配或最相似的部分。
- 对象识别的主要任务是在源图像中寻找所有指定的对象。

本章主要介绍图像的特征统计、特征提取、特征描述和景物匹配等方面的基础知识。

16.2 图像的特征统计

这里以图像的灰度直方图为例介绍图像的特征统计。

在对图像进行处理前，对图像整体（当然也可以是局部）的灰度分布情况作一些分析了解，通常是很有好处的。对图像的灰度分布进行分析的重要手段就是建立灰度直方图。

灰度直方图是灰度级的函数，它表示图像中具有某种灰度级的像素的个数，反映了图像中某种灰度出现的频率。也就是说，灰度直方图是对图像的所有像素的灰度分布按灰度值的大小显示灰度值出现频度的统计图。

16.2.1 直方图的表示

通常，灰度直方图的横轴用来表示灰度值，纵轴用来表示频度。频度是具有某一灰度值（或属于某一子区域的灰度值）的像素在图像中出现的次数。

1. 灰度值的频度

举例说明。假设有一幅4×4的8灰度级图像，图像数据及其灰度直方图分别如图16-1左侧和图16-1右侧所示。

1	3	2	7
2	4	0	2
1	2	3	5
2	4	4	2

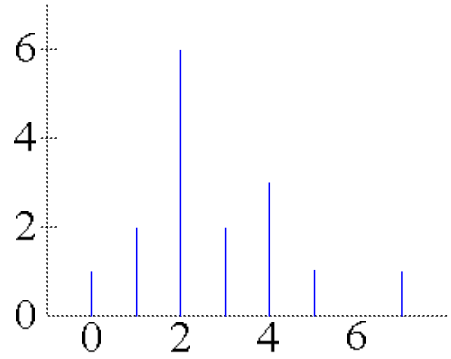


图16-1 一个8灰度级图像及其灰度直方图

因为根据图像数据，灰度值为0, 1, 2, 3, 4, 5, 6, 7的像素的频度分别是1, 2, 6, 2, 3, 1, 0, 1，因此在灰度刻度的0~7处分别作一条以该灰度值对应的频度值为长度的直线即可完成该图像的灰度直方图。

2. 灰度值子区间的频度

举例说明。假设有一个矩阵包含一张图像的灰度值（如图16-2左侧所示）。

41	36	194	141	241	133	250
223	146	225	173	106	129	100
173	204	179	199	31	165	23
197	34	170	100	207	177	245
22	0	203	159	179	83	209

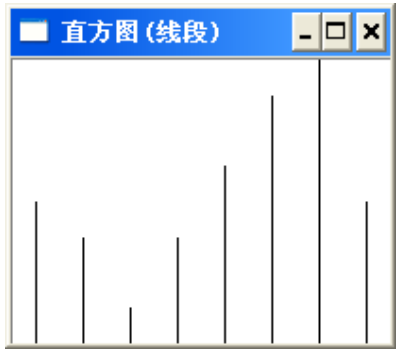


图16-2　一个数字矩阵及其直方图

灰度值的范围包含256个值，可以将这个范围均匀分割成若干子区域，如8个均匀子区域，即 $[0,256)=[0,32)\cup[32,64)\cup\ldots\cup[224,256)$ ①。

将灰度值范围均匀分割成若干子区域以后，统计灰度值属于每一个子区域的像素数目。

采用这一方法来统计上面的数字矩阵，可以得到如图16-2右侧所示的直方图（ x 轴表示子区域， y 轴表示属于各个子区域的像素个数）。

① 为了介绍方便，本章只考虑将灰度值范围均匀分割成若干子区域的情况。

3#. 一个参考程序

下面给出具体实现的一个参考程序，该程序用于计算一幅单通道图像的灰度直方图。

```
// 计算一个单通道图像的直方图(源图像, 直方图, 大小, 范围)
void MyCalcHist(Mat image, Mat &hist, int size, vector<float> range)
{   float a = range[0], b = range[1]; // 下界, 上界
    if(b < a) swap(a, b);
    Mat1f H(1, size, 0.0); // 直方图初始值为0
    for(auto v : Mat1f(image))
    {   int i = (v - a) * size / (b - a); // 当前像素所在子区域
        ++H(i); // 相应子区域的像素数目增加1
    }
    hist = H; // 记录结果
}
```

16.2.2 OpenCV中的直方图操作

在OpenCV中，直方图是对数据集合的统计，并将统计结果分布于一系列预定义的子区域中。

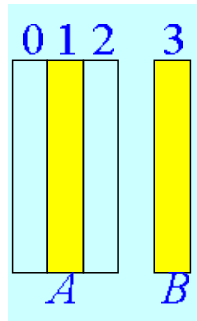
这里的数据不仅仅是灰度值，也可以是其他能有效描述图像特征的数据（如梯度，方向等），还可以是多个特征空间中的数据（例如，多通道图像，多个图像等）。

下面先举例说明一些细节，然后介绍OpenCV中的相关函数。

1. 一些细节

举例说明。假设有2个输入图像*A*和*B*，通道数分别是3和1，只关心图像*A*的1号通道和图像*B*，如图16-3所示。

- 通道编号：OpenCV在计算直方图时，将各输入图像的所有通道从0开始统一编号。例如，对于上述*A*和*B*，各通道统一编号为{0, 1, 2, 3}。
- channels：使用通道编号选取各图像中待统计的通道。例如，对于上述*A*和*B*，若channels = {1, 3}，则只统计第1通道和第3通道。
- ranges：指定每个通道的取值范围，是一组成对出现的float数，第i对数表示第i维的下界和上界。例如，对于上述*A*和*B*，若ranges = {0, 256, 10, 70}，则第1维的取值范围是[0, 256)，第3维的取值范围是[10, 70)。
- histSize：指定每个特征空间子区域的数目，即将第i维的整个区域均匀分割成histSize[i]个子区域。例如，对于上述*A*和*B*，若histSize = {8, 4}，则第1维的取值范围[0, 256)均匀分割成8个子区域，第3维的取值范围[10, 70)均匀分割成4个子区域。



channels = {1, 3} // COI
ranges = {0, 256, 10, 70}
histSize = {8, 4}

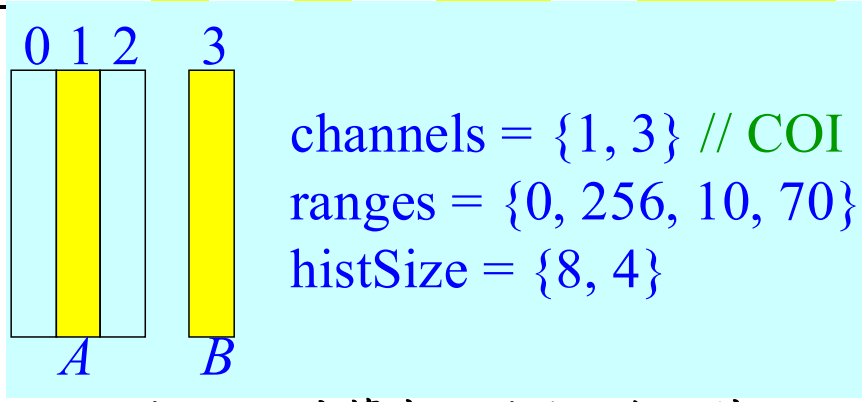


图16-3　计算直方图的几个细节

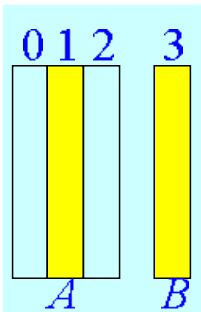
2. 相关函数

【函数原型】[①]void calcHist(InputArrayOfArrays images, const std::vector<int> &channels, InputArray mask, OutputArray hist, const std::vector<int> &histSize, const std::vector<float> &ranges);

【功能】计算图像的直方图。

【参数】

- images：输入图像数组，通常使用vector<Mat>对象。各输入图像的位深度和大小必须一致，通道数可以不同，位深度必须是CV_8U或CV_32F。
- channels：各图像中待统计的通道，用从0开始的统一编号选取。
- mask：确定需要统计输入图像中的哪些像素。
- hist：计算出来的直方图。
- histSizes：直方图各维的大小。
- ranges：直方图中各维的取值范围，是一组成对出现的float数，每一对float数表示一个范围。



channels = {1, 3} // COI
ranges = {0, 256, 10, 70}
histSize = {8, 4}

① 这里只给出了将灰度值范围均匀分割成若干子区域的情况。

16.2.3　OpenCV绘制直方图举例

1. 用线段方式绘制直方图

下列程序使用线段方式绘制一幅单通道图像的灰度直方图，保存在文件cvv.hpp中。

```
// 用黑白图像绘制一个单通道图像的灰度直方图
// 线段: 源图像(单通道图像), 结果图像, 直方图线段数, 取值范围
void drawHistLines(Mat src, Mat &dst, int size,
                   const std::vector<float> &range = {0, 256})
{   Mat1f hist; // 直方图
    // 计算直方图(图像集, 通道, 掩码, 直方图, 各维大小, 范围)
    calcHist(std::vector{src}, {0}, Mat(), hist, {size}, range);
    // 使直方图高度和结果图像高度一致
    normalize(hist, hist, dst.rows - 1, 0, NORM_INF);
    dst.setTo(255); // 结果图像使用白色背景
    float dx = (float)dst.cols / size; // 子区域宽度
    for(int i = 0; i < size; ++i)
    {   // 绘制时可假设原点在下方, 完成后垂直翻转
        int x = i * dx + dx / 2, y = hist(i);
        Point pt1(x, 0), pt2(x, y); // 下端点和上端点
        line(dst, pt1, pt2, {0}); // 黑色线段
    }
    flip(dst, dst, 0); // 结果图像垂直翻转
}
```

2. 使用示例

程序运行结果如图16-4所示。

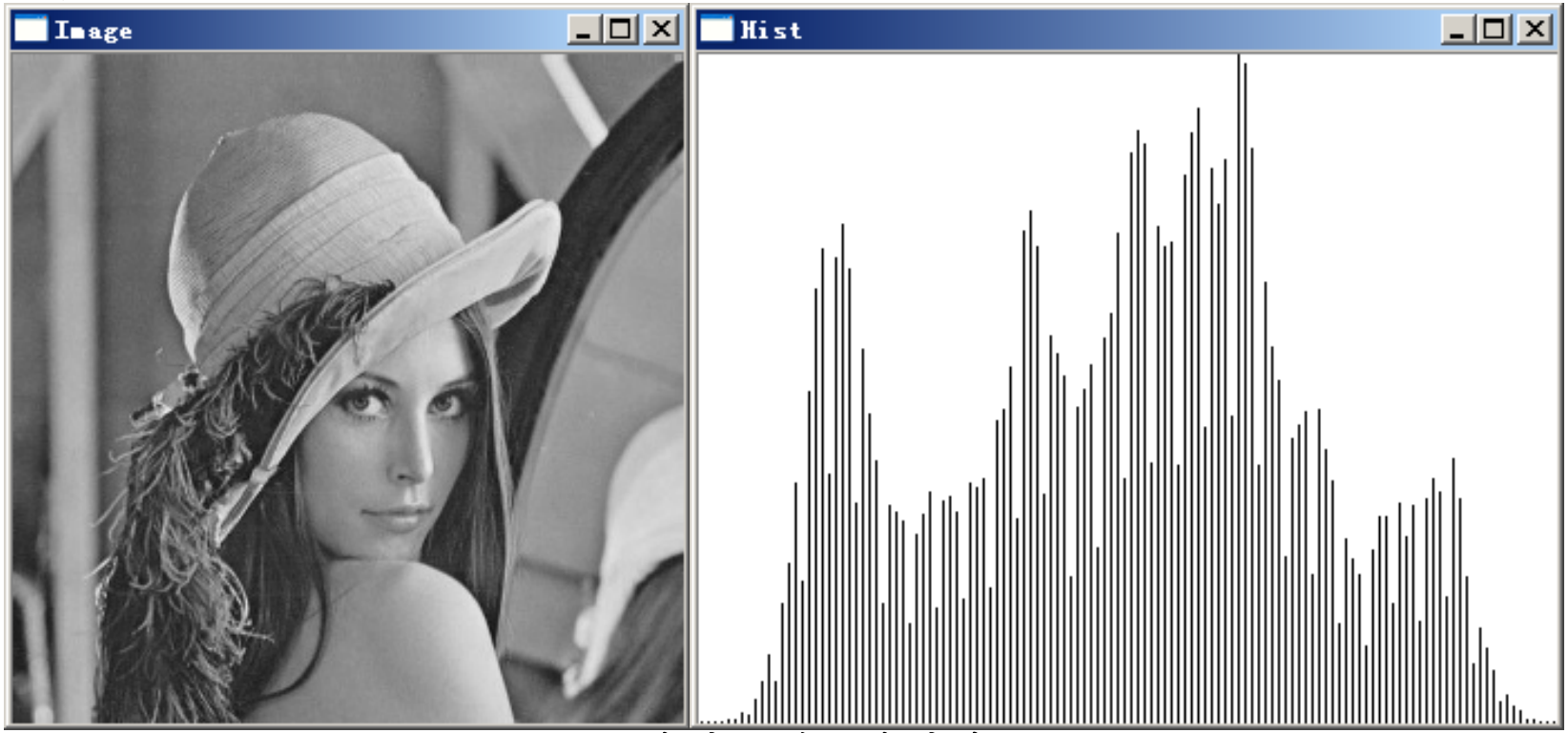


图16-4　灰度图像的灰度直方图

```
// DrawHistLines.Cpp
#include"cvv.hpp"
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 载入灰度图像
    if(X.empty()) return -1; // 载入图像失败
    imshow("Image", X); // 显示源图像
    Mat Y(300, 384, CV_8U); // 直方图是黑白的
    drawHistLines(X, Y, 128); // 生成直方图，线段
    imshow("Hist", Y);
    waitKey();
}
```

16.2.4# 基于直方图的图像增强

直方图均衡化是一种基于直方图的图像增强方法。

1. 直方图均衡化的目的

直方图均衡化也是一种对于灰度的变换，它与前述章节介绍的线性变换有所不同。线性变换是把像素的灰度分布扩展到较大灰度域的一种灰度变换，而直方图均衡化既要求尽量扩展灰度的分布域，又需要努力使每一个灰度级上的频度尽可能一致。

这种力求使灰度分布域上的频度趋近一致化的努力是有道理的。因为频度趋于一致的图像使人感觉色调沉稳、安定，在许多情况下这意味着图像质量“好”。

2. 直方图均衡化的处理

直方图均衡化的处理是按照下列方法完成的。

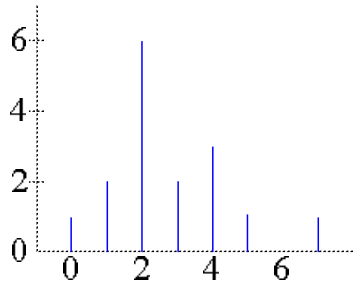
- 计算归一化因子：$s = \bar{g} / |f|$，其中 $|f|$ 是源图像 f 的像素总数，整数 $\bar{g}$ 是最大灰度级，通常为255。
- 计算源图像的直方图 $H[0..\bar{g}]$。
- 计算直方图积分：$H'[i] = \sum_{j=0}^{i} H[j]$ ①。
- 直方图积分归一化：计算公式为 $H'[i] = H'[i] \times s$（舍入取整）。
- 采用 H' 作为查询表对源图像的灰度 $f(x,y)$ 进行变换得到结果图像的灰度 $g(x,y)$，即 $g(x,y) = H'[f(x,y)]$。

例如，对于图16-1所示的8灰度级图像数据，实际操作如图16-5所示。需要注意，这里只有8个灰度级，共16个像素，所以直方图只有8个元素，且归一化因子为 $s = 7/16$。

① 此处 OpenCV 手册中的描述有误。这里的步骤描述依据 equalizeHist()函数的源程序整理。

1	3	2	7
2	4	0	2
1	2	3	5
2	4	4	2

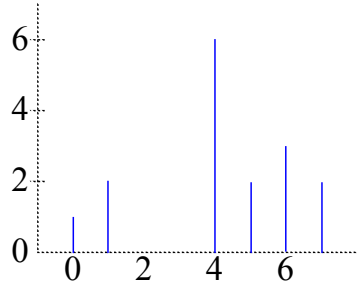
源图像



源直方图

1	5	4	7
4	6	0	4
1	4	5	7
4	6	6	4

结果图像



结果直方图

灰度级（源亮度）	0	1	2	3	4	5	6	7
源直方图	1	2	6	2	3	1	0	1
直方图积分	1	3	9	11	14	15	15	16
归一化（结果亮度）	0	1	4	5	6	7	7	7
结果直方图	1	2	0	0	6	2	3	2

图16-5　直方图均衡化的实际操作

【注】直方图均衡化后灰度等级数很可能会减少。容易看出，对于图16-1所示的图像数据，直方图均衡化后灰度等级由7个变成了6个。

3. 一种实现方法

```
void MyEqualizeHist(Mat src, Mat &dst, int gray = 256)
{   // 源图像, 结果图像, 灰度级数, 使用大小相同的单精度字节图像
    int total = src.total(); // 像素总数
    float g = gray , s = (g - 1) / total; // 归一化因子
    // 计算直方图
    Mat1f X = src; // 源图像, 计算时使用单精度数图像
    Mat1f H; // 直方图
    calcHist(vector{X}, {0}, Mat(), H, {gray}, {0, g});
    for(int i = 1; i < g; ++i) // 直方图积分
        H(i) += H(i - 1);
    H *= s; // 直方图积分归一化
    Mat1f Y(src.size()); // 结果图像, 计算时使用单精度数图像
    for(int i = 0; i < total; ++i) // 灰度变换
        Y(i) = H(X(i));
    Y.convertTo(dst, src.type()); // 转换结果图像(舍入取整)
}
```

4. OpenCV中的相关函数

【函数原型】void equalizeHist(InputArray src, OutputArray dst);

【功能】对灰度图像进行直方图均衡化。

【参数】src和dst分别是源图像和结果图像，是大小相同的单通道字节图像。

5. 应用举例

下列程序演示了直方图均衡化对图像的影响。运行结果如图16-6所示。

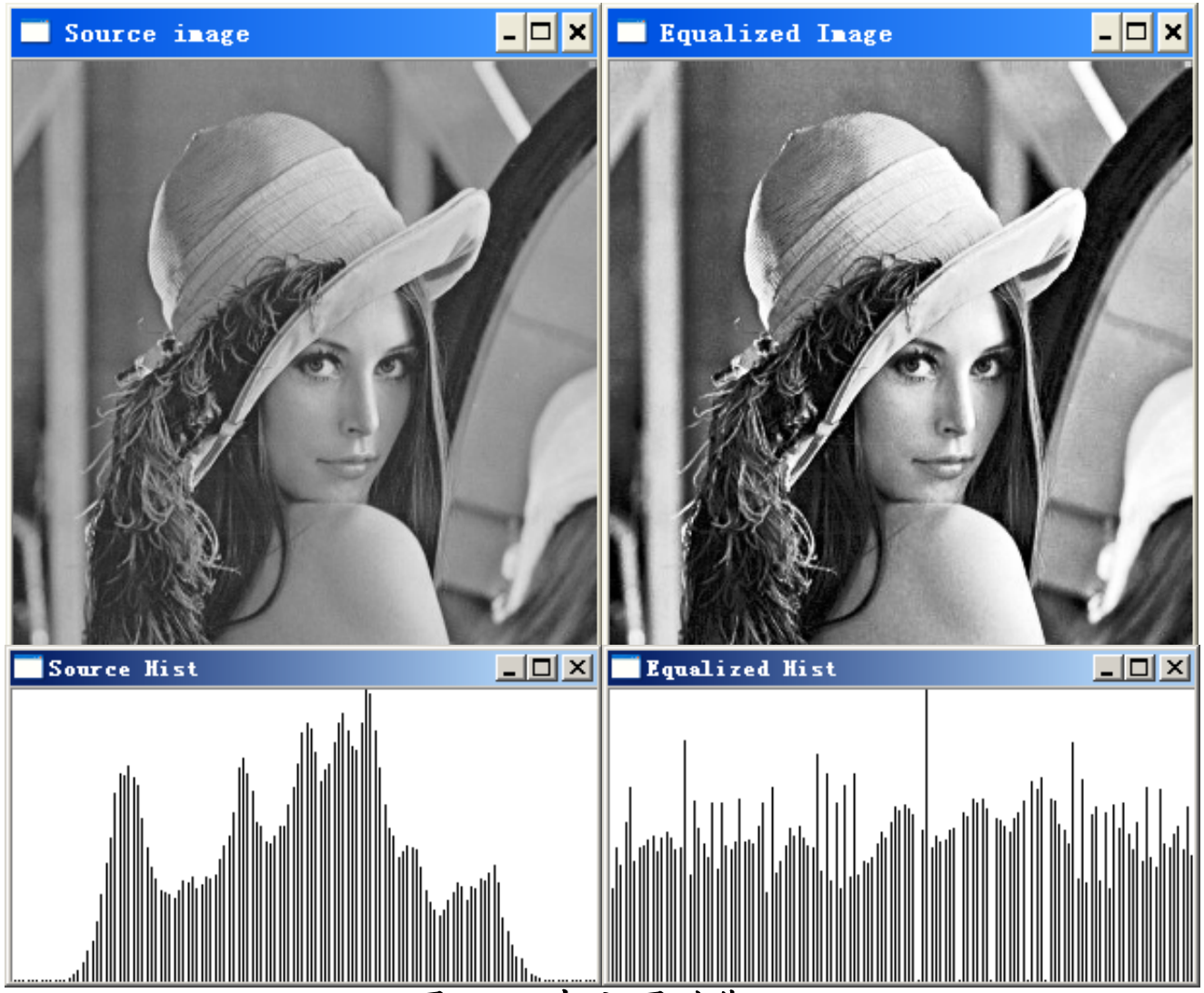


图16-6 直方图均衡化

```
// EqualizeHist.Cpp
#include"cvv.hpp"
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 加载源图像
    if(X.empty()) return -1;
    imshow("Source image", X); // 显示源图像
    Mat Y(150, 300, CV_8U); // 直方图是黑白的
    drawHistLines(X, Y, 128); // 灰度直方图
    imshow("Source Hist", Y); // 显示灰度直方图
    equalizeHist(X, X); // 应用直方图均衡化
    imshow("Equalized Image", X); // 显示结果图像
    drawHistLines(X, Y, 128); // 灰度直方图
    imshow("Equalized Hist", Y); // 显示灰度直方图
    waitKey();
}
```

16.3 图像的二值化

16.3.1 二值图像

- 多值图像是指具有多个灰度级的单色图像，例如一张黑白照片。
- 二值图像是指只有黑白两个灰度级的图像。
- 图像的二值化是指将一幅多值图像转换成一幅二值图像。
- 为了达到压缩图像数据、突出图像特征、便于进行图形识别等目标，需要进行图像的二值化。

16.3.2 二值化变换

1. 二值化变换处理的方法

图像的二值化变换处理的方法很简单。在源图像的灰度区间$[I_{\min}, I_{\max}]$设定一个阈值t（$I_{\min} \le t \le I_{\max}$），然后将图像中所有灰度值大于$t$的像素的灰度值都改为1，其余像素的灰度值都改为0。

```
Y.setTo(0);
Y.setTo(v, X > t); // v代表1
```

2. 正确选择阈值的重要性

必须注意，多值图像经二值化处理后会丢失源图像中的许多信息，处理得不好的图像有可能使得源图像面目全非，这样就完全失去了图像二值化的意义。因为图像二值化的目标是要在尽可能多地保留源图像特征的前提下舍弃冗余信息。实现这一目标的关键在于正确选择阈值。

3. 自适应阈值

自适应阈值使用非固定阈值，计算方法比较简单。首先将源图像使用 $n \times n$ 的模板进行平滑处理（通常是均值模糊或高斯模糊，n 是奇数），得到新数组 T。然后将数组 T 中的每个元素都减去一个预先指定的值 C（$T(x,y) = T(x,y) - C$），将新的 $T(x,y)$ 作为源像素 (x,y) 的阈值。

在进行二值化时，如果源像素 (x,y) 的灰度值大于 $T(x,y)$，则将该像素的灰度值改为1，否则，将该像素的灰度值改为0。

```
Y.setTo(0);
blur(X, T, {n, n}); // 或GaussianBlur(X, T, {n, n}, 0);
T = T - C;
Y.setTo(v, X > T); // v代表1
```

16.3.3 直方图对选择阈值的帮助

灰度直方图可以对正确选择阈值提供帮助。

1. 二值倾向比较明显的情况

对于那些原本具有二值倾向的灰度图像来说，问题比较容易解决。所谓具有二值倾向的图像是指图像的背景色与前景色截然不同，轮廓非常分明的图像，例如用扫描仪扫描得到的一幅工程图图像，灰白色的背景上深色的线条和文字很容易区分。这种图像的灰度直方图呈现出两峰一谷的特征，如图16-7所示。这时若取谷底处的灰度值作为阈值，一般可以得到较好的结果。

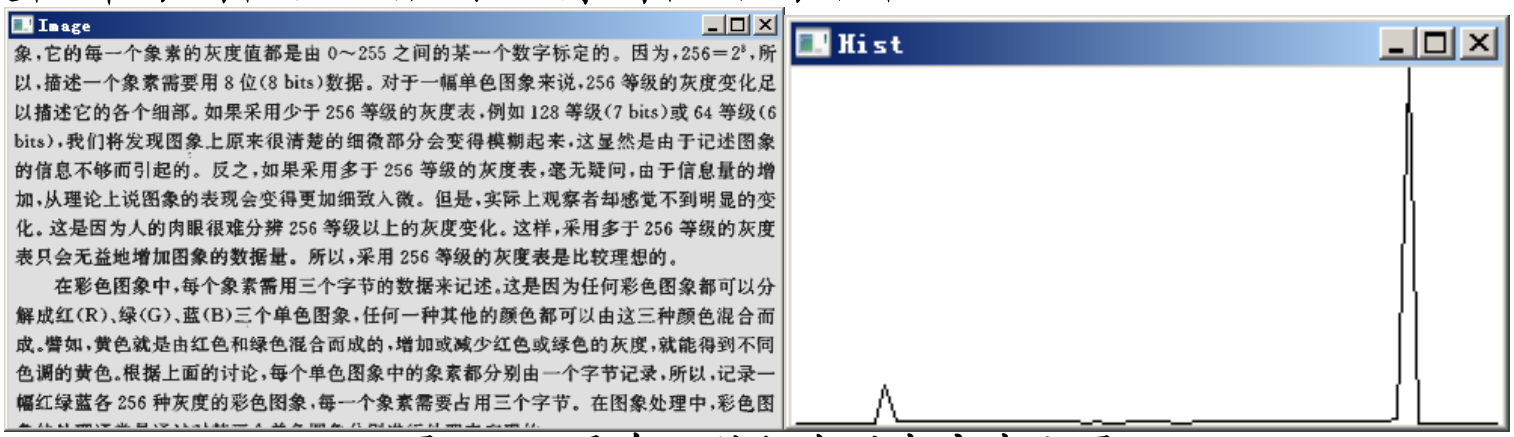


图16-7 具有二值倾向的灰度直方图

2. 二值倾向不明显的情况

（1）先进行预处理。对于二值倾向不明显的图像，在需要进行二值化处理时，应先对图像进行预处理，以增强图像的轮廓等特征，这样可保证在二值化处理后能更多地保留源图像的特征。

二值化的预处理中通常还包括消除噪音的处理。消除噪音的方法有多种，对于不同类型的噪音需要用不同的方法处理。

（2）选取、调整阈值。预处理后，在灰度直方图的谷底处（尽管此时峰与谷可能不太明显）取一个值作为阈值，如果用此阈值生成的二值图像效果不佳，可以适当修改阈值，重新进行二值化处理。

阈值调整的原则是：若二值图像失去的特征过多（如轮廓严重残缺等），则适当增大阈值；若二值图像有较多的冗余信息（如轮廓线太粗，噪音点较多等），则应适当减小阈值。经过两、三次调整后，一般能够获得较满意的二值化结果。

3#. 借助直方图选取阈值的一种参考实现

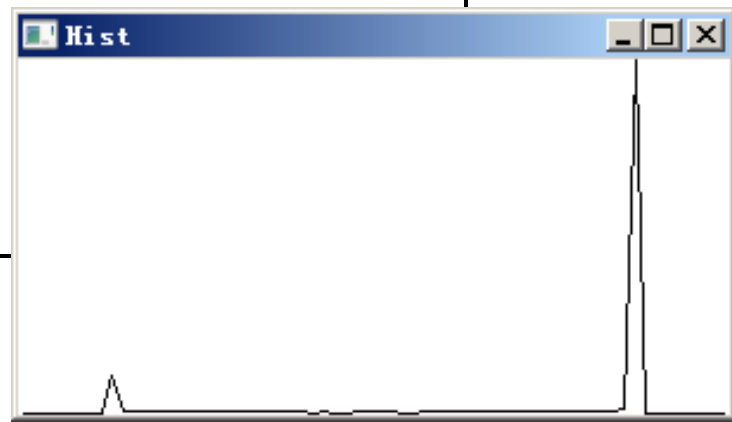
（1）峰和谷。

```
bool isPeak(Mat1f hist, int i) // 峰
{   if(i <= 0) return hist(i) > hist(i + 1);
    if(i >= hist.total() - 1) return hist(i) > hist(i - 1);
    return hist(i) > max(hist(i - 1), hist(i + 1));
}
```

```
bool isValley(Mat1f hist, int i) // 谷
{   if(i <= 0) return hist(i) < hist(i + 1);
    if(i >= hist.total() - 1) return hist(i) < hist(i - 1);
    return hist(i) < min(hist(i - 1), hist(i + 1));
}
```

（2）选取阈值。

```
int getThreshold(Mat1b src, int histSize = 64) // 源图像, 直方图大小
{   Mat1f hist; // 直方图
    // 计算直方图(图像集, 通道, 掩码, 直方图, 各维大小, 范围)
    calcHist(vector{src}, {0}, Mat(), hist, {histSize}, {0, 256});
    int f = 0; // 首峰
    while(f < histSize and not isPeak(hist, f)) ++f;
    int p = f; // 最高峰
    for(int i = f + 1; i < histSize; ++i)
        if(isPeak(hist, i) and hist(i) > hist(p)) p = i;
    int q = f; // 次高峰
    for(int i = f + 1; i < histSize; ++i)
        if(i != p and isPeak(hist, i) and hist(i) > hist(q)) q = i;
    // 两峰之间的最低处用作阈值
    if(q < p) swap(p, q);
    int t = p;
    for(int i = p + 1; i < q; ++i)
        if(hist(i) < hist(t)) t = i;
    return (t / histSize + 0.5) * 256;
}
```



16.3.4 OpenCV中常用的阈值种类

表16-1列出了OpenCV中常用的阈值种类。

表16-1 OpenCV中的阈值种类

阈值种类	计算公式	操作结果
二值化阈值 THRESH_BINARY	$d_i = \text{iif}(s_i > t, v, 0)$	二值图像，高亮度改为指定值v
反二值化阈值 THRESH_BINARY_INV	$d_i = \text{iif}(s_i > t, 0, v)$	反二值图像，低亮度改为指定值v
截断阈值 THRESH_TRUNC	$d_i = \text{iif}(s_i > t, t, 0)$	二值图像，高亮度改为阈值
0阈值 THRESH_TOZERO	$d_i = \text{iif}(s_i > t, s_i, 0)$	灰度图像，过滤低亮度像素
反0阈值 THRESH_TOZERO_INV	$d_i = \text{iif}(s_i > t, 0, s_i)$	灰度图像，过滤高亮度像素

16.3.5 相关函数与应用举例

1. 固定阈值

【函数原型】double threshold(InputArray src, OutputArray dst, double thresh, double value, int type);

【功能】对数组元素进行固定阈值操作。

【参数】

- src：源数组，位深度为CV_8U或CV_32F的单通道数组。
- dst：输出数组，大小和类型与src一致，必要时重建。
- thresh：阈值。
- value：指定二值化和反二值化使用的替代值。
- type：阈值类型（见表16-1）。

【返回值】thresh。

2. 自适应阈值

【函数原型】void adaptiveThreshold(InputArray src, OutputArray dst, double value, int adaptiveMethod, int thresholdType, int blockSize, double C);

【功能】对数组元素进行自适应阈值操作。

【参数】

- src：源数组，位深度为CV_8U或CV_32F的单通道数组。
- dst：输出数组，大小和类型与src一致，必要时重建。
- value：指定二值化和反二值化使用的替代值。
- adaptiveMethod：计算自适应阈值时使用的的模糊方法，只能是ADAPTIVE_THRESH_MEAN_C（均值模糊）或ADAPTIVE_THRESH_GAUSSIAN_C（高斯模糊）。
- thresholdType：阈值类型，只能是THRESH_BINARY（二值化）或THRESH_BINARY_INV（反二值化）。
- blockSize：模板大小，不超过31的奇数。
- C：预先指定的减数。

3. 应用举例

下列程序用于演示灰度图像的二值化，给出了使用固定阈值和自适应阈值以及不同阈值对二值化效果的影响。程序运行结果如图16-8所示。

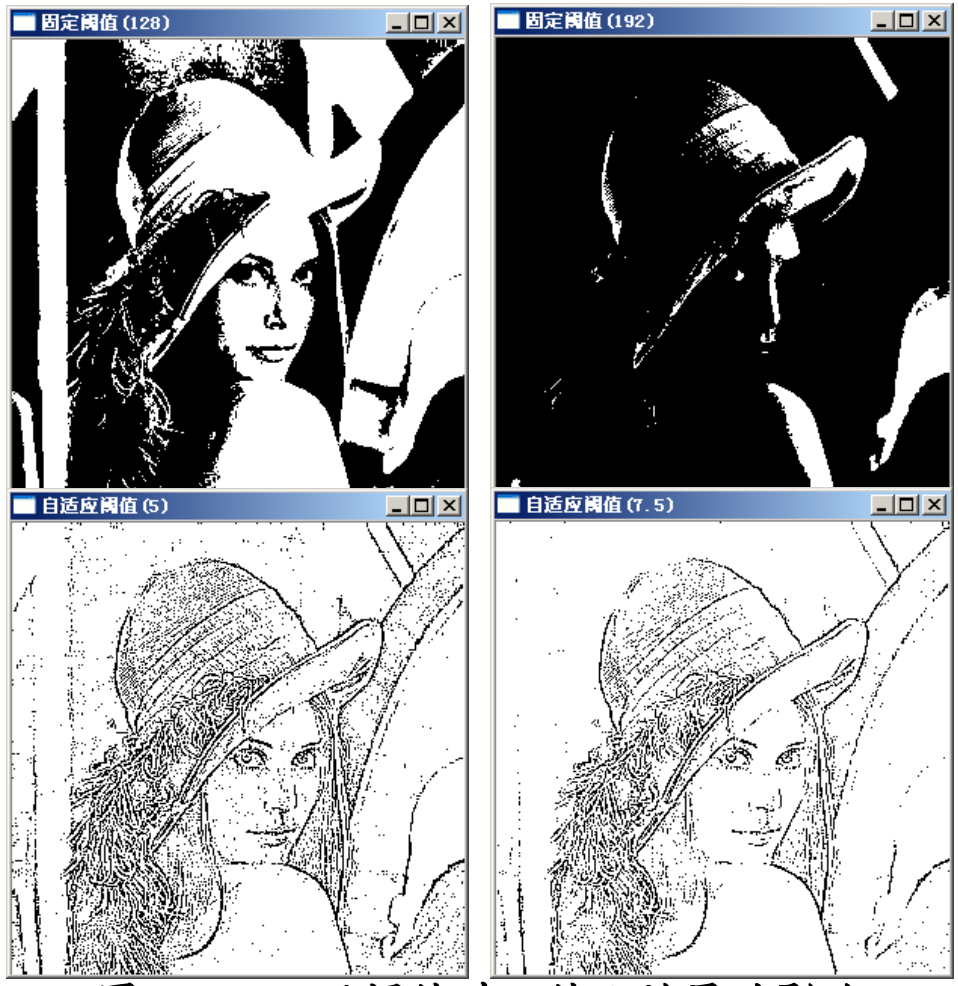


图16-8　不同阈值对二值化效果的影响

```
// Threshold.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 载入灰度图像
    if(X.empty()) return -1;
    Mat Y; // 结果图像
    threshold(X, Y, 128, 255, THRESH_BINARY);
    imshow("固定阈值(128)", Y);
    threshold(X, Y, 192, 255, THRESH_BINARY);
    imshow("固定阈值(192)", Y);
    adaptiveThreshold(X, Y, 255, ADAPTIVE_THRESH_MEAN_C,
                      THRESH_BINARY, 3, 5);
    imshow("自适应阈值(5)", Y);
    adaptiveThreshold(X, Y, 255, ADAPTIVE_THRESH_MEAN_C,
                      THRESH_BINARY, 3, 7.5);
    imshow("自适应阈值(7.5)", Y);
    waitKey();
}
```

16.4 图像的特征提取

这里以边缘检测为例介绍图像的特征提取。

图像的边缘点是指图像中周围像素灰度有阶跃变化或屋顶变化的那些像素点，即灰度值导数较大或极大的地方。

边缘检测可以大幅度减少数据量，并且剔除不相关信息，保留图像的重要结构属性。

16.4.1 二值图像的边缘检测

这里介绍一种比较简单的针对二值图像的边缘检测方法。约定背景像素为黑色，图形像素为白色。

1. 几个基本术语

- 白像素。灰度值大于0的像素为白像素，否则为黑像素。
- 孤立点。周围8个像素都是黑像素的白像素。
- 端点。周围8个像素中正好有1个白像素的白像素。
- 内部点。周围4个像素（上下左右）都是白像素的白像素。
- 边缘像素。不是内部点的白像素，即周围4个像素中至少有一个黑像素的白像素。

【注】边界外像素当作黑像素处理。

2. 二值图像边缘检测的基本方法

首先将边缘图像的所有像素初始化为黑像素，然后在源图像中沿图像扫描方向搜索，检查像素是否是边缘像素。将检测到的边缘像素记录到边缘图像中，即将边缘图像中的相应像素改为白像素。

3. 举例

对于图16-9左侧所示的二值图像，使用上述基本方法检测到的边缘如图16-9右侧所示。

1	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	0	0	1	1	1	1
1	0	0	0	1	1	1
0	0	0	0	0	1	1
0	1	0	0	0	0	1

1	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	0	0	1	0	0	1
1	0	0	0	1	0	1
0	0	0	0	0	1	1
0	1	0	0	0	0	1

图16-9　边缘检测

4#. 二值图像边缘检测方法的一种参考实现

```
bool isEdge(const Mat1b &im, int x, int y) // 检查边缘像素
{   if(im(y, x) <= 0) return false; // 黑像素
    return at(im, y - 1, x) <= 0 or at(im, y + 1, x) <= 0 or // 上，下
           at(im, y, x - 1) <= 0 or at(im, y, x + 1) <= 0; // 左，右
}
```

```
Mat1b getEdges(Mat1b src) // 获取边缘，源图像是二值图像
{   Mat1b dst(src.size(), 0); // 结果图像，初始为黑色
    for(int x = 0; x < src.cols; ++x) // 记录边缘像素
        for(int y = 0; y < src.rows; ++y)
            if(isEdge(src, x, y)) dst(y, x) = 255;
    return dst;
}
```

16.4.2 灰度图像的边缘检测

通常使用Canny边缘检测算法对灰度图像进行边缘检测。Canny边缘检测算法是John F. Canny于1986年开发出来的一个多级边缘检测算法，被很多人认为是边缘检测的最优算法。

1. Canny边缘检测算法的主要步骤

（1）消除噪音。通常使用高斯平滑（可选）。

（2）计算梯度幅值和方向（两次锐化）。通常使用Sobel滤波器的一阶差分模板。

- 运用一对卷积（用 $*$ 表示）。假设被作用图像为 I，则

$$g_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} * I, \quad g_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} * I$$

- 使用下列公式计算梯度幅值和方向。

$$g = \sqrt{g_x^2 + g_y^2}, \quad \theta = \arctan(g_y / g_x)$$


```
Sobel(src, dst, ddepth, 1, 0, 3);
Sobel(src, dst, ddepth, 0, 1, 3);
```

（3）非极大值抑制。排除非边缘像素，仅保留一些细线条（候选边缘）。首先将梯度方向近似到4个可能角度之一（一般是0、45、90和135度，若角度不属于区间 $[0°,180°)$ 则使用其对称角度），然后检查当前像素的梯度幅值是否是近似梯度方向上的极大值，如果不是，则不保留该像素。例如，假设当前像素 (x,y) 的近似梯度方向为45度，使用 3×3 模板，如果 (x,y) 的梯度幅值不是 $(x-1,y-1)$、(x,y) 和 $(x+1,y+1)$ 中最大的，则不保留 (x,y)。

（4）滞后阈值。滞后阈值需要两个阈值，称为高阈值和低阈值，也可称为大阈值和小阈值。

- 若像素的梯度幅值小于低阈值，则排除该像素。
- 若像素的梯度幅值大于高阈值，则保留该像素。
- 若像素的梯度幅值在低阈值和高阈值之间，则该像素只有在连接到（注意，不是邻接）一个梯度幅值大于高阈值的像素时才保留[①]。
- Canny推荐的高低阈值比在2:1到3:1之间。

① 也可以理解为，若像素的梯度幅值在两个阈值之间，则该像素只有在邻接到一个已经保留的像素时才保留。这样理解需要多次检测。

（5）剔除孤立点。

【说明】上述算法得到的结果与OpenCV中的Canny()函数得到的结果稍有差异，Canny()函数做了更细致的处理。

2. 相关函数

【函数原型】void Canny(InputArray image, OutputArray edges, double threshold1, double threshold2);

【功能】采用Canny算法做边缘检测。

【参数】

- image和edges：输入图像和输出图像（保存边缘），都是灰度图像。
- threshold1和threshold2：小阈值和大阈值。

3. 举例说明

下列程序用于演示Canny边缘检测，给出了使用不同阈值的边缘检测效果（高低阈值比为3:1）。程序运行结果如图16-10所示。

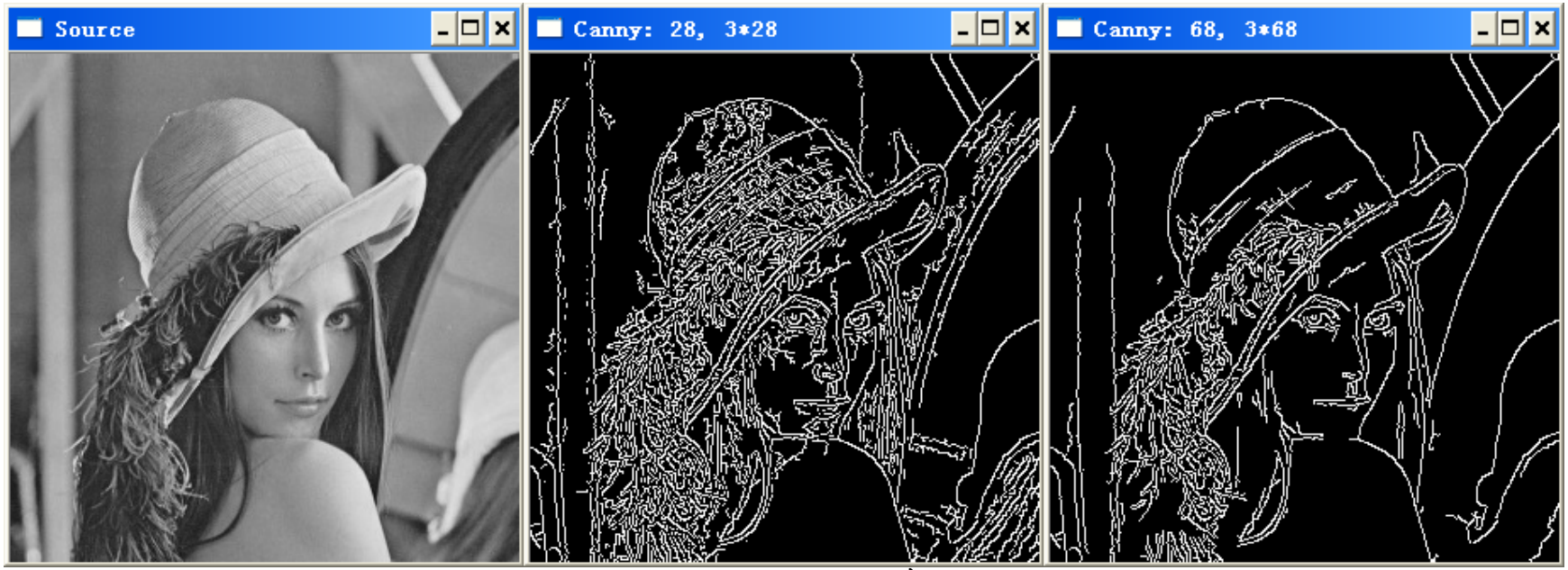


图16-10　Canny边缘检测

```
// Canny.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg", 0); // 载入灰度图像
    if(X.empty()) return -1; // 载入图像失败
    imshow("Source", X); // 显示源图像
    Mat Y; // 结果图像
    Canny(X, Y, 28, 3 * 28); // 阈值1 = 28, 阈值2 = 3×28
    imshow("Canny: 28, 3*28", Y);
    Canny(X, Y, 68, 3 * 68); // 阈值1 = 68, 阈值2 = 3×68
    imshow("Canny: 68, 3*68", Y);
    waitKey();
}
```

16.4.3# Canny边缘检测算法的一种参考实现

1. 主要步骤

（1）各主要步骤相关函数。

```
// 主要步骤
void grads(Mat src, Mat &mag, Mat &ang); // 计算梯度幅值和方向
void restrain(Mat &mag, Mat ang); // 非极大值抑制
void lagThreshold(Mat1f &im, double low, double high); // 使用滞后阈值
void eliminated(Mat1f &im); // 剔除孤立点
```

（2）主算法描述。

```
void MyCanny(Mat src, Mat &dst, double low, double high)
{   // 消除噪音(可选)
    Mat1f mag, ang; // 梯度幅值和方向
    grads(src, mag, ang); // 计算梯度幅值和方向
    restrain(mag, ang); // 非极大值抑制
    lagThreshold(mag, low, high); // 使用滞后阈值
    eliminated(mag); // 剔除孤立点
    mag.convertTo(dst, src.type()); // 记录结果
}
```

2. 主要步骤的实现方法

（1）计算梯度幅值和方向。

```
// 计算梯度幅值和方向(两次锐化)
void grads(Mat src, Mat &mag, Mat &ang)
{   Mat1f dx, dy; // x差分和y差分
    Sobel(src, dx, dx.type(), 1, 0, 3); // x差分
    Sobel(src, dy, dy.type(), 0, 1, 3); // y差分
    cartToPolar(dx, dy, mag, ang, true); // 梯度幅值和方向
}
```

（2）非极大值抑制。

```
// 非极大值抑制
bool isMax(Mat1f mag, Mat1f ang, int x, int y); // 是否极大值
```

```
void restrain(Mat &mag, Mat ang) // 非极大值抑制
{   Mat1b mask(mag.size()); // 标识非极大值
    for(int x = 0; x < mag.cols; ++x)
        for(int y = 0; y < mag.rows; ++y)
            mask(y, x) = not isMax(mag, ang, x, y); // 非极大值
    mag.setTo(0, mask); // 非极大值抑制
}
```

```
bool isMax(Mat1f mag, Mat1f ang, int x, int y) // 是否极大值
{   float mg = mag(y, x), ag = ang(y, x);
    switch((int)round(ag / 45) % 4)
    {   case 0: // 0度
            return mg >= max(at(mag, y, x - 1), at(mag, y, x + 1));
        case 1: // 45度
            return mg >= max(at(mag, y - 1, x - 1), at(mag, y + 1, x + 1));
        case 2: // 90度
            return mg >= max(at(mag, y - 1, x), at(mag, y + 1, x));
        case 3: // 135度
            return mg >= max(at(mag, y - 1, x + 1), at(mag, y + 1, x - 1));
    }
    return false;
}
```

（3）使用滞后阈值。

```
// 使用滞后阈值
void extend(Mat1f &im, int x, int y); // 扩展白像素
```

```
void lagThreshold(Mat1f &im, double low, double high) // 使用滞后阈值
{   im.setTo(0, im < low); // 剔除低亮度像素
    im.setTo(255, im > high); // 保留高亮度像素
    for(int x = 0; x < im.cols; ++x) // 扩展白像素
        for(int y = 0; y < im.rows; ++y)
            if(im(y, x) >= 255) extend(im, x, y);
    im.setTo(0, im < 255); // 剔除剩余的中间亮度像素
}
```

```
void extend(Mat1f &im, int x, int y) // 扩展白像素
{   for(int u = x - 1; u <= x + 1; ++u)
        for(int v = y - 1; v <= y + 1; ++v)
        {   if(v == y and u == x) continue; // 不考虑(x, y), 避免重复
            float g = at(im, v, u);
            if(g > 0 and g < 255) // 只扩展中间亮度像素
                at(im, v, u) = 255, extend(im, u, v); // 改成白像素并扩展
        }
}
```

（4）剔除孤立点

```
// 剔除孤立点
bool isolated(Mat1f im, int x, int y); // 孤立点
```

```
void eliminated(Mat1f &im) // 剔除孤立点
{   for(int x = 0; x < im.cols; ++x)
        for(int y = 0; y < im.rows; ++y)
            if(isolated(im, x, y)) im(y, x) = 0;
}
```

```
bool isolated(Mat1f im, int x, int y) // 孤立点
{   if(im(y, x) <= 0) return false;
    for(int u = x - 1; u <= x + 1; ++u)
        for(int v = y - 1; v <= y + 1; ++v)
        {   if(u == x and v == y) continue; // 不考虑(x, y)
            if(at(im, v, u) > 0) return false;
        }
    return true;
}
```

16.5 图像的特征描述

这里以轮廓的检测与描述为例介绍图像的特征描述。

轮廓检测（也可称为轮廓追踪）是一种非常有用的图像解析工具，主要用于形状分析和对象的检测与识别。虽然边缘检测算法能够获得边缘像素，但是不能检测出图像中有哪些轮廓线条，每个轮廓具体有哪些组成部分，所以需要有专门的轮廓检测方法。

轮廓检测算法能够检测出图像中的轮廓以及每个轮廓的组成部分，并将这些轮廓按照某种方式组织，便于遍历每个轮廓以及了解轮廓之间的关系。

16.5.1 轮廓线追踪的基本方法

轮廓线追踪是点阵图形的矢量化和模式识别等领域中常用的一种手法，其目的是沿着图形的等色区域的边界搜索，将搜索到的边界线（轮廓线）上的点记录在点序列中，结果是一个点序列，表示一条轮廓线。

遍历追踪算法是一种思路比较简单的轮廓线追踪算法，是轮廓线追踪最基本的方法，适用于二值图像的轮廓线追踪。

1. 基本步骤

这里约定背景像素为黑色，图形像素为白色。

（1）从源图像中获得边缘图像。可以使用对二值图像进行边缘检测的基本方法。

（2）寻找轮廓线追踪的起点。在边缘图像中沿图像扫描方向搜索，检查像素为白还是黑。把最先检测到的白像素 P_1 作为轮廓线追踪的起点。该起点像素是全画面中最左上位置的白像素。将 P_1 加入到轮廓线的点序列中，并在边缘图像中将 P_1 改为黑像素。

（3）寻找下一像素。按照某种指定的追踪顺序（例如图16-11所示的追踪顺序）依次检查 P_1 周围的8个像素是否是白像素，将首次检测到的白像素设为 P_2，加入到轮廓线的点序列中，并在边缘图像中将 P_2 改为黑像素。

7	6	5
8	P_1	4
1	2	3

图16-11　一种追踪顺序

（4）完成一条轮廓线的追踪。如果已经检测出 P_{n-1}，则将 P_{n-1} 作为模板中心像素，按同样的方法搜索 P_n，直到不能搜索到白像素为止。点序列 $P_1, P_2, \ldots, P_n$ 就是一条轮廓线（P_n 和 P_1 相邻表示追踪到一条封闭的轮廓线）。

（5）下一轮廓线的追踪。在边缘图像中继续寻找搜索起始点，进行下一条轮廓线的追踪。直到边缘图像扫描完毕为止。

2. 举例

对于图16-9所示的二值图像，使用遍历追踪算法按照图16-11所示的追踪顺序可以得到4条轮廓线。第1条轮廓线有5个像素，坐标分别是(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)；第2条轮廓线有4个像素，坐标分别是(1, 0), (2, 0), (3, 0), (4, 0)；第3条轮廓线有12个像素，坐标分别是(2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (6, 5), (6, 4), (6, 3), (6, 2), (5, 2), (4, 2), (3, 2)；第4条轮廓线只有1个像素，坐标是(1, 6)。注意，已经追踪到的像素在边缘图像中改成了黑像素，不会重复追踪。

1	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	0	0	1	0	0	1
1	0	0	0	1	0	1
0	0	0	0	0	1	1
0	1	0	0	0	0	1

7	6	5
8	P_1	4
1	2	3

16.5.2 轮廓线的符号描述

这里以轮廓线的存储和组织方式为例来介绍轮廓线的符号描述。

1. 轮廓线的存储

通常选用下列两种方法存储轮廓线。

（1）直接存储点序列。例如，从图16-9所示的二值图像中追踪到的4条轮廓线分别存储为

- C0 = {(0, 0), (0, 1), (0, 2), (0, 3), (0, 4)}
- C1 = {(1, 0), (2, 0), (3, 0), (4, 0)}
- C2 = {(2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (6, 5), (6, 4), (6, 3), (6, 2), (5, 2), (4, 2), (3, 2)}
- C3 = {(1, 6)}

1	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	0	0	1	0	0	1
1	0	0	0	1	0	1
0	0	0	0	0	1	1
0	1	0	0	0	0	1

7	6	5
8	P_1	4
1	2	3

（2）压缩存储点序列。压缩水平、垂直和对角部分，只保留线段的端点。例如，从图16-9所示的二值图像中追踪到的4条轮廓线分别存储为

- C0 = {(0, 0), (0, 4)}
- C1 = {(1, 0), (4, 0)}
- C2 = {(2, 2), (6, 6), (6, 2), (3, 2)}
- C3 = {(1, 6)}

1	1	1	1	1	0	0
1	0	0	0	0	0	0
1	0	1	1	1	1	1
1	0	0	1	0	0	1
1	0	0	0	1	0	1
0	0	0	0	0	1	1
0	1	0	0	0	0	1

7	6	5
8	P_1	4
1	2	3

2. 轮廓线的提取和组织方式

（1）只提取最外层轮廓。这种方式只提取最外层的轮廓线，并将这些轮廓线根据最上顶点按照由下往上，从右到左的顺序组织成一个双向列表。例如，对于图16-12左侧所示的二值图像，这种方式只提取轮廓线C0和C1，并组织成右侧第1行所示的双向列表。

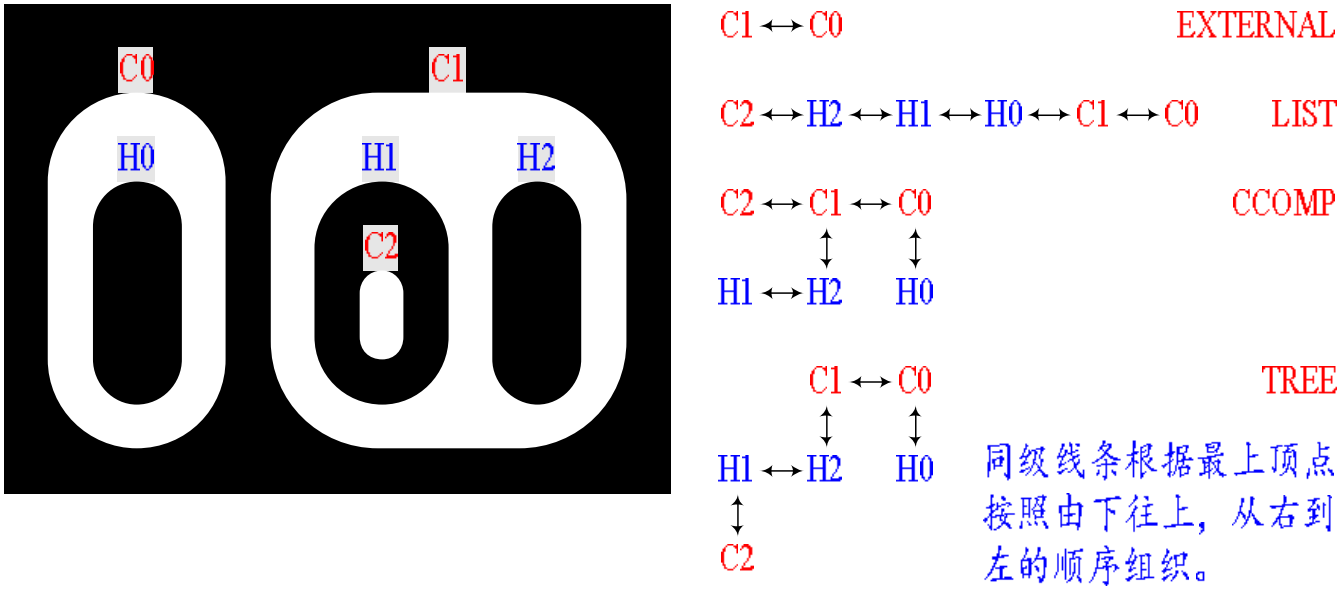
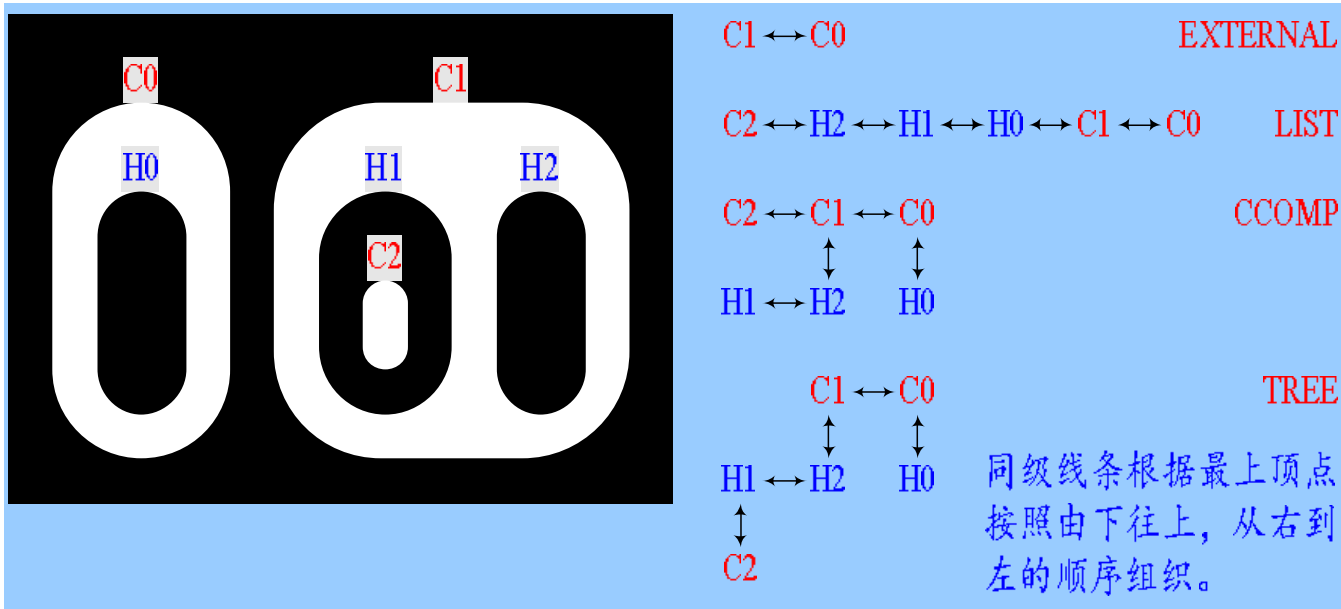


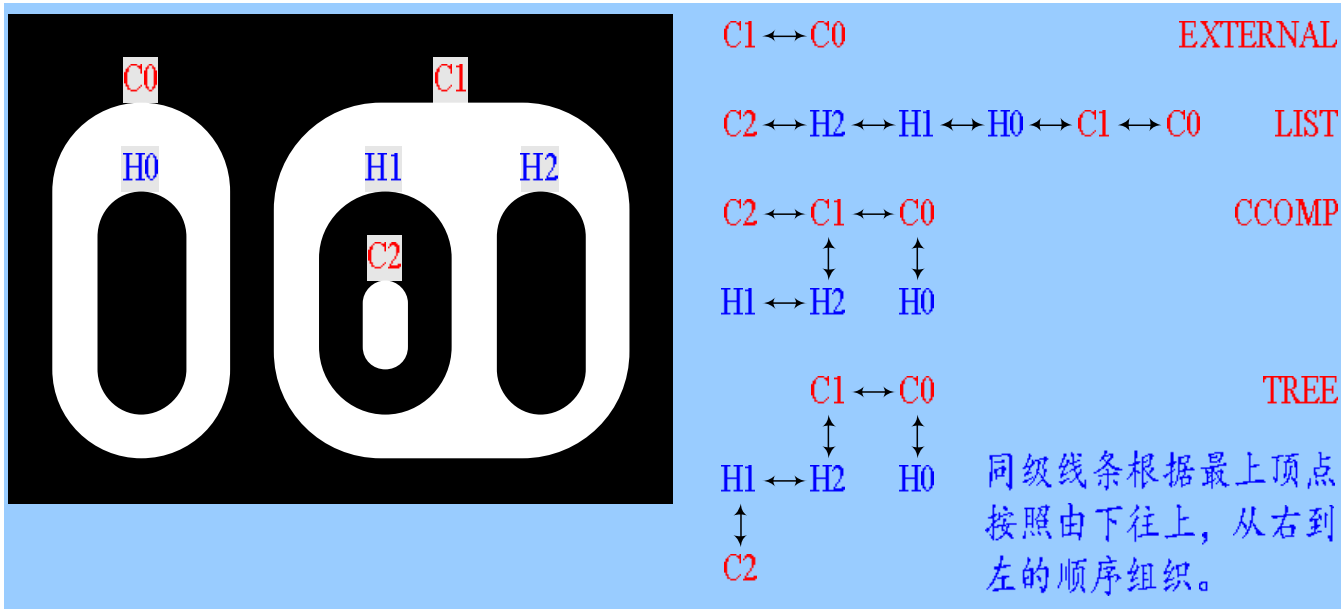
图16-12　轮廓的提取和组织方式

（2）列表方式。这种方式提取所有轮廓线，并将这些轮廓线根据最上顶点按照由下往上，从右到左的顺序组织成一个双向列表。例如，对于图16-12左侧所示的二值图像，这种方式提取所有轮廓线，并组织成右侧第2行所示的双向列表。

（3）双层结构方式。这种方式提取所有轮廓线，并将这些轮廓线组织成双层结构，上层是外围边界，下层是洞的边界，同级的轮廓线根据最上顶点按照由下往上，从右到左的顺序组织。例如，对于图16-12左侧所示的二值图像，这种方式提取所有轮廓线，并组织成右侧第3行所示的双层结构。



（4）树结构方式。这种方式提取所有轮廓线，并重构这些轮廓线的等级，用儿子兄弟的方式将这些轮廓线组织成树结构，同级的轮廓线根据最上顶点按照由下往上，从右到左的顺序组织。例如，对于图16-12左侧所示的二值图像，这种方式提取所有轮廓线，并组织成右侧第4行所示的树结构。



16.5.3 OpenCV中的相关函数

1. findContours

【函数原型】void findContours(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method);

【功能】从二值图像中提取轮廓，并获得轮廓的层次信息。

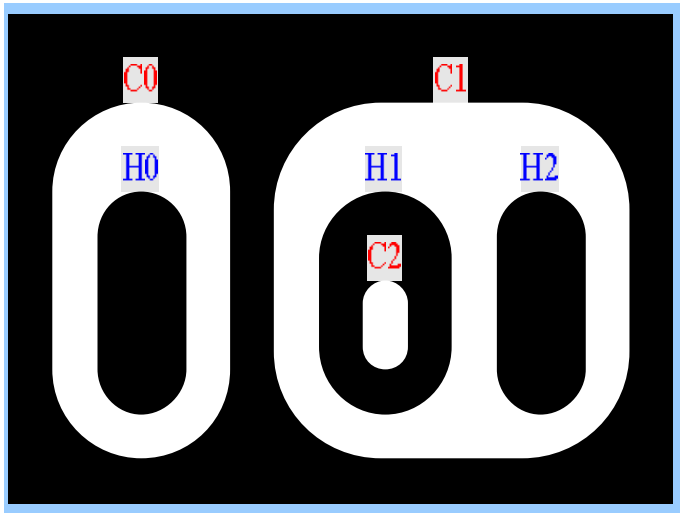
【参数】

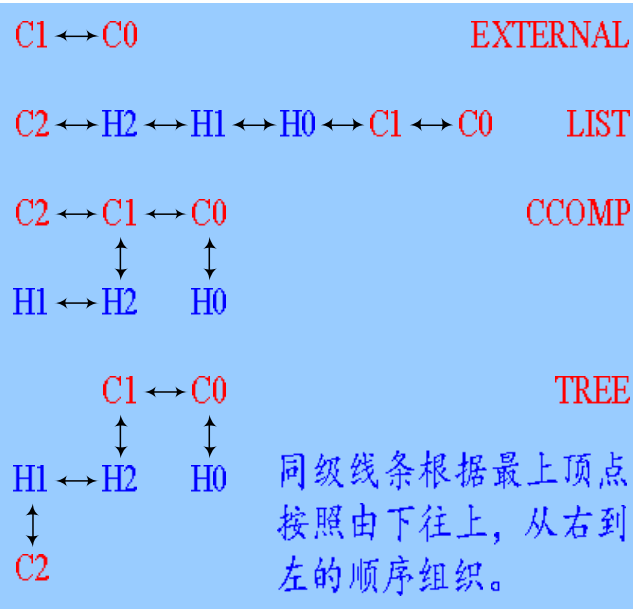
- image：单通道二值图像，即将非0元素当成1，0元素保留为0。
- contours：检测到的轮廓，每个轮廓就是一个点序列。
- hierarchy：标记轮廓间的层次关系（另一种findContours()无此参数）。
- mode：轮廓的提取和组织方式，可选RETR_EXTERNAL（只提取最外层轮廓）、RETR_LIST（提取所有轮廓，组织成列表）、RETR_CCOMP（提取所有轮廓，组织成双层结构）和RETR_TREE（提取所有轮廓，组织成树结构）。
- method：轮廓线的存储方法，通常选用CHAIN_APPROX_NONE（直接输出点序列）或CHAIN_APPROX_SIMPLE（压缩水平、垂直和对角部分）。

【说明】举例说明层次关系。对于轮廓i，hierarchy[i]包含4个元素，分别代表轮廓i的下一轮廓（上或左）、前一轮廓（右或下）、第一个儿子轮廓和双亲轮廓，负数代表相应轮廓不存在。

例如，如果将图16-12中的各轮廓组织成树结构，则

- hierarchy[C1] = [C0, -1, H2, -1] // 0
- hierarchy[H2] = [H1, -1, -1, C1] // 1
- hierarchy[H1] = [-1, H2, C2, C1] // 2
- hierarchy[C2] = [-1, -1, -1, H1] // 3
- hierarchy[C0] = [-1, C1, H0, -1] // 4
- hierarchy[H0] = [-1, -1, -1, C0] // 5





2. drawContours

【函数原型】void drawContours(InputOutputArray image, InputArrayOfArrays contours, int contourIdx, const Scalar &color, int thickness = 1);

【功能】在图像中绘制轮廓线或填充轮廓区域。

【参数】

- image：用于绘制轮廓的图像。
- contours：所有轮廓。
- contourIdx：指定绘制哪条轮廓，负数默认表示绘制所有轮廓。
- color：轮廓的颜色。
- thickness：轮廓线条的粗细程度。负数表示绘制轮廓线填充内外层轮廓之间的区域（由选用的轮廓线判断内外层关系）。

16.5.4 举例说明

下列程序演示了对一幅二值图像进行轮廓检测的效果，该程序绘制出了检测到的所有轮廓（区分了外部轮廓和内部轮廓），并演示了填充效果。程序运行结果如图16-13所示。

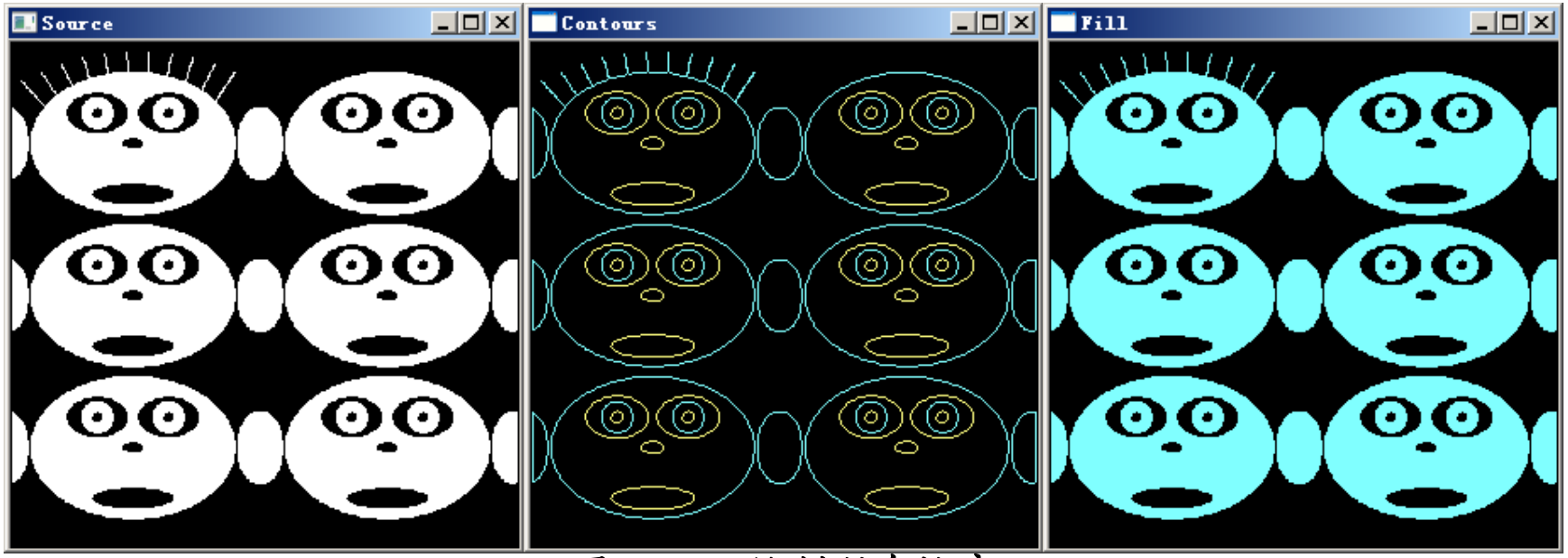


图16-13 绘制所有轮廓

```
// Contours.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main()
{   Mat X = imread("Contours.bmp", 0); // 载入二值图像
    if(X.empty()) return -1; // 载入图像失败
    imshow("Source", X); // 显示源图像
    // 查找轮廓
    vector<vector<Point>> cont; // 轮廓线数组
    vector<Vec4i> hierarchy; // 轮廓的层次信息
    int mode = RETR_CCOMP; // 提取所有轮廓，组织成双层结构
    int method = CHAIN_APPROX_SIMPLE; // 压缩水平，垂直和对角
    // 查找轮廓(源图像，轮廓线数组，层次信息，组织方式，存储方法)
    findContours(X, cont, hierarchy, mode, method);
```

```
// 绘制轮廓线
Mat Y(X.size(), CV_8UC3); // 结果图像(彩色图像)
Y.setTo(0); // 设置背景(黑色)
// 外部轮廓颜色, 内部轮廓颜色
Scalar outer(255, 255, 128), inner(128, 255, 255);
// 从0开始依次选取下一个外部轮廓
for(int i = 0; i >= 0; i = hierarchy[i][0])
{   // 绘制外层轮廓线
    drawContours(Y, cont, i, outer); // 只绘制轮廓i
    // 绘制i的所有儿子(内层轮廓线)
    // 从首儿子开始依次选取下一个儿子
    for(int j = hierarchy[i][2]; j >= 0; j = hierarchy[j][0])
        drawContours(Y, cont, j, inner); // 只绘制轮廓j
}
imshow("Contours", Y); // 显示结果图像
```

```
    // 填充轮廓
    Y.setTo(0); // 重新绘制
    drawContours(Y, cont, -1, outer, FILLED); // 所有轮廓，填充
    imshow("Fill", Y); // 显示结果图像
    waitKey();
}
```

16.6 图像中的景物匹配

这里以模板匹配为例介绍图像中的景物匹配。

16.6.1 模板匹配的基本原理

模板匹配是一项在源图像中寻找与模板图像最匹配（相似）部分的技术。当然，模板图像的宽度和高度都不超过源图像。

为了确定匹配区域，需要滑动模板图像和源图像进行比较。通过滑动，即模板图像块一次移动一个像素（自上而下，从左往右从上往下）。在每一个位置，都通过匹配算法进行一次度量计算，确定模板图像和源图像的特定区域有多么相似。计算出匹配结果后，从源图像中选取最匹配或最相似的区域。如图16-14所示。

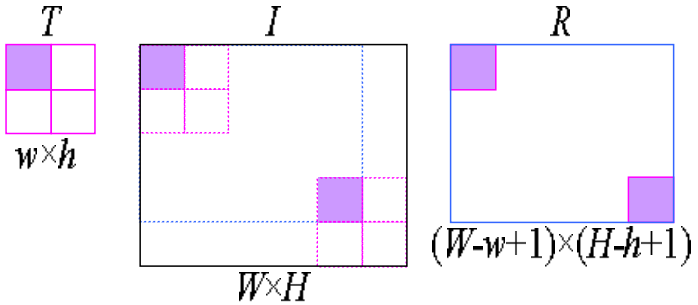
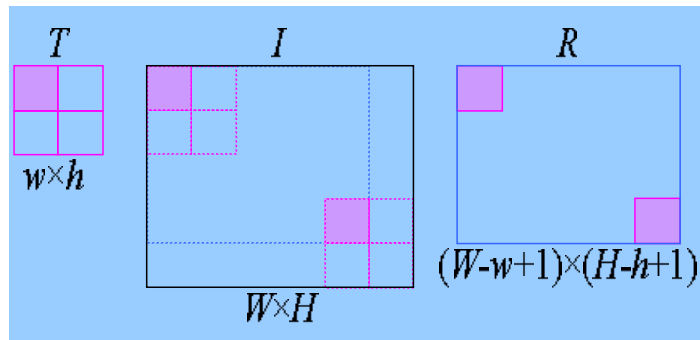


图16-14 模板匹配的基本原理

16.6.2 OpenCV支持的匹配算法

用 I 表示源图像，宽度和高度分别是 W 和 H，用 T 表示模板图像，宽度和高度分别是 w 和 h，用 R 表示匹配结果，是一个实数矩阵，宽度和高度分别是 $W-w+1$ 和 $H-h+1$，$R(x,y)$ 表示模板图像与源图像中 $[x,x+w)\times[y,y+h)$ 区域的匹配或相似程度。如图16-14所示。

为了方便，用 $(u,v)\in T$ 表示 $(u,v)\in[0,w)\times[0,h)$。



1. 差平方匹配

$$R(x,y)=\sum_{(u,v)\in T}(I(x+u,y+v)-T(u,v))^2$$

$R(x,y)$越小，则模板图像与源图像的相应区域越匹配。

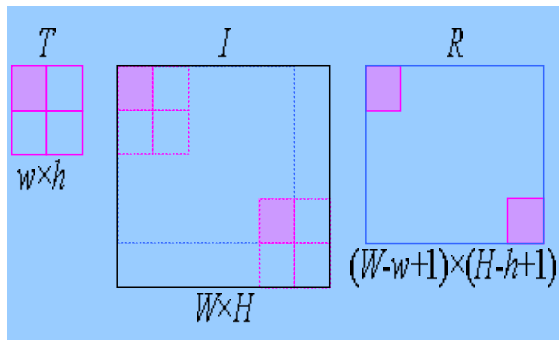
```
Mat S = I({x, y, w, h});
Mat I2 = S - T;
R(y, x) = I2.dot(I2);
```

2. 归一化的差平方匹配

$$R(x,y)=\frac{\sum_{(u,v)\in T}(I(x+u,y+v)-T(u,v))^2}{\sqrt{\sum_{(u,v)\in T}(I(x+u,y+v))^2}\sqrt{\sum_{(u,v)\in T}(T(u,v))^2}}$$

$R(x,y)$越小，则模板图像与源图像的相应区域越匹配。

```
Mat S = I({x, y, w, h});
Mat I2 = S - T;
R(y, x) = I2.dot(I2);
R(y, x) /= norm(S) * norm(T);
```



3. 相关匹配

这类方法采用模板和图像间的乘法操作，所以较大的数表示匹配程度较高，0表示最坏的匹配效果。

$$R(x,y)=\sum_{(u,v)\in T} I(x+u,y+v)\,T(u,v)$$

$R(x,y)$越大，则模板图像与源图像的相应区域越匹配。

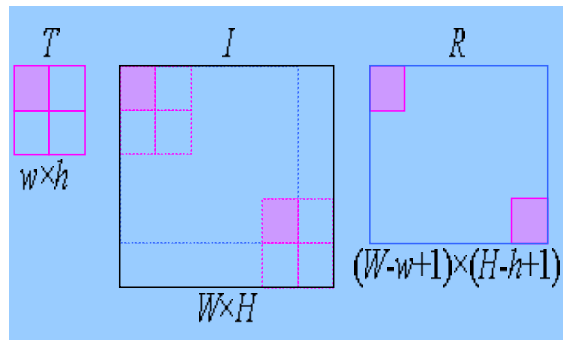
```
Mat S = I({x, y, w, h});
R(y, x) = S.dot(T);
```

4. 归一化的相关匹配

$$R(x,y)=\frac{\sum_{(u,v)\in T} I(x+u,y+v)\,T(u,v)}{\sqrt{\sum_{(u,v)\in T}(I(x+u,y+v))^2}\sqrt{\sum_{(u,v)\in T}(T(u,v))^2}}$$

$R(x,y)$越大，则模板图像与源图像的相应区域越匹配。

```
Mat S = I({x, y, w, h});
R(y, x) = S.dot(T);
R(y, x) /= norm(S) * norm(T);
```



5. 相关系数匹配

这类方法将模版对其均值的相对值与图像对其均值的相对值进行匹配，1表示完美匹配，-1表示糟糕的匹配，0表示没有任何相关性（随机序列）。

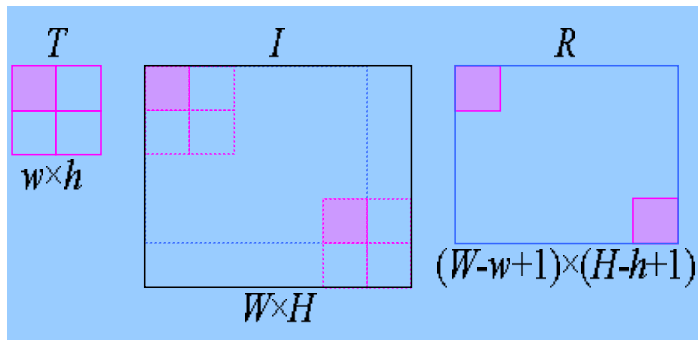
$$R(x,y)=\sum_{(u,v)\in T} I'(x+u,y+v)\,T'(u,v)$$

其中

$$T'(u,v)=T(u,v)-\text{avg}\{T(s,t):(s,t)\in T\}$$
$$I'(x+u,y+v)=I(x+u,y+v)-\text{avg}\{I(x+s,y+t):(s,t)\in T\}$$

$R(x,y)$越大，则模板图像与源图像的相应区域越匹配。

```
Mat S = I({x, y, w, h});
Mat I2 = S - mean(S), T2 = T - mean(T);
R(y, x) = I2.dot(T2);
```



6. 归一化的相关系数匹配

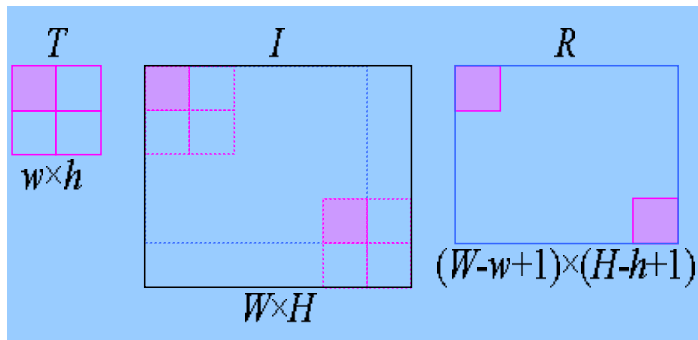
$$R(x,y)=\frac{\sum_{(u,v)\in T} I'(x+u,y+v)\,T'(u,v)}{\sqrt{\sum_{(u,v)\in T}(I'(x+u,y+v))^2}\sqrt{\sum_{(u,v)\in T}(T'(u,v))^2}}$$

其中

$$I'(x+u,y+v)=I(x+u,y+v)-\operatorname{avg}\{I(x+s,y+t):(s,t)\in T\}$$
$$T'(u,v)=T(u,v)-\operatorname{avg}\{T(s,t):(s,t)\in T\}$$

$R(x,y)$越大，则模板图像与源图像的相应区域越匹配。

```
Mat S = I({x, y, w, h});
Mat I2 = S - mean(S), T2 = T - mean(T);
R(y, x) = I2.dot(T2);
R(y, x) /= norm(I2) * norm(T2);
```



16.6.3 OpenCV中的模板匹配函数

【函数原型】void matchTemplate(InputArray image, InputArray templ, OutputArray result, int method);

【功能】比较模板和重叠的图像区域。

【参数】

- image和templ：源图像和模板图像，同为字节图像或单精度数图像。
- result：结果矩阵，一个通道的单精度数矩阵。
- method：匹配方法。可选
  - TM_SQDIFF（差平方匹配）
  - TM_SQDIFF_NORMED（归一化的差平方匹配）
  - TM_CCORR（相关匹配）
  - TM_CCORR_NORMED（归一化的相关匹配）
  - TM_CCOEFF（相关系数匹配）
  - TM_CCOEFF_NORMED（归一化的相关系数匹配）

【说明】函数完成比较后，可以通过使用minMaxLoc寻找全局最小值（差平方匹配）或者最大值（相关匹配和相关系数匹配）。

```
void minMaxLoc(InputArray src, double *minVal, double *maxVal = 0,
    Point *minLoc = 0, Point *maxLoc = 0, InputArray mask = noArray());
```

16.6.4 举例说明

下列程序演示了对一幅彩色图像进行模板匹配的效果。该程序使用差平方匹配算法进行模板匹配，源图像中与模板最匹配的区域用一个蓝色矩形标记。程序运行结果如图16-15所示。

图16-15 模板匹配

```
// MatchTemplate.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat I = imread("lena.jpg"); // 源图像
    Mat T = imread("Template.jpg"); // 模板图像
    imshow("模板", T); // 显示模板图像
    Mat R; // 结果矩阵
    matchTemplate(I, T, R, TM_SQDIFF); // 模板匹配(差平方匹配)
    Point minLoc; // 保存最小值位置，即矩形起点
    minMaxLoc(R, 0, 0, &minLoc); // 最小值位置
    Rect rect{minLoc, T.size()}; // 矩形(起点, 大小)
    rectangle(I, rect, {255, 0, 0}); // 绘制矩形(蓝色)
    imshow("匹配", I); // 显示匹配结果
    waitKey();
}
```

16.7 练习题

16.7.1 基础知识题

1. 请用线段方式绘制下列8灰度级图像的灰度直方图。

1	3	6	4	1	4
6	6	2	0	1	1
1	3	1	3	3	6
3	4	7	4	6	1
4	6	5	4	6	7

2. 请对下列8灰度级图像进行直方图均衡化。

1	3	6	4	1
4	6	6	2	0
1	1	1	3	1
3	3	6	3	4

3. 假设在一幅10×10的8灰度级图像中，各灰度级的频度如下表所示，请对该图像进行直方图均衡化（给出每个源亮度对应的结果亮度）。

灰度级	0	1	2	3	4	5	6	7
频度	12	24	26	15	12	5	3	3

4. 请使用二值图像边缘检测的基本方法和轮廓线追踪的遍历追踪算法检测下列二值图像中的轮廓线。

0	0	1	1	1	1	1
0	0	0	0	0	0	1
0	0	0	0	1	0	1
0	0	0	1	1	0	1
0	0	1	1	1	0	1
0	1	1	1	1	0	0
1	1	1	1	1	0	0

16.7.2 程序设计题

1. 使用OpenCV编写一个程序，该程序对一幅灰度图像进行直方图均衡化，要求分别显示源图像和均衡化以后的图像。

2. 使用OpenCV编写一个程序，该程序对一幅灰度图像进行二值化变换，要求分别显示源图像和二值化以后的图像。其中二值化阈值为127，高亮度改为255。

3. 使用OpenCV编写一个程序，该程序对一幅灰度图像进行Canny边缘检测，要求分别显示源图像和检测到的边缘。其中小阈值为50，大阈值为150，内核大小为3。

4. 使用OpenCV编写一个程序，该程序首先使用Canny算法检测边缘，然后从源图像中复制出边缘像素。注意，源图像是彩色图像，边缘检测时需转换成灰度图像，结果图像也是彩色图像。

5. 使用OpenCV编写一个程序，该程序对一幅彩色图像（例如当前目录中的lena.jpg）使用指定的模板图像（例如当前目录中的Template.jpg）进行模板匹配。要求使用分别使用OpenCV支持的6种匹配算法分别进行模板匹配，源图像中与模板最匹配的区域分别使用一个指定颜色的矩形标记。

16.7.3 阶段实习题

1. 将使用线段方式绘制直方图的函数改写成用直方块方式绘制直方图。方法是在每一个子区间绘制一个高度为频度的矩形。

2. 将使用线段方式绘制直方图的函数改写成用折线方式绘制直方图。方法是将线段方式中每条线段的上端点连接起来。

3. 使用OpenCV编写一个程序，该程序完成在源图像中使用特殊颜色直接标记出轮廓的任务。

4. 对于一幅二值倾向比较明显的图像，请借助直方图找到比较合适的阈值完成该图像的二值化。相关函数的使用请参阅OpenCV手册。

5. 小块污损区域的检测。使用OpenCV编写一个小块污损区域检测程序。在一幅彩色图像中有少量明显的小块污损区域，使用该程序检测出这些污损区域，并将每个污损区域用一个矩形框标记出来。

6. 静态手势识别。以石头、剪子、布三种手势为例。通过一个很普通的摄像头捕捉手势，使用一个程序识别该手势到底是石头，还是剪子，还是布。要求不能对手势的位置和方向等属性进行规定，也不能对周围环境提出太明显的要求。请使用OpenCV编写这个程序。