# 图像二值化

方法一：最原始的方法，就是对RGB彩色图像灰度化以后，扫描图像的每个像素值，值小于127的将像素值设为0(黑色)，值大于等于127的像素值设为255(白色)。该方法的优点是计算速度快。缺点是因为阀值是127固定的，完全不考虑图像的像素分布情况与像素值特征。

方法二：该方法则相对于第一个方法改进的地方是：计算像素的平均值K，扫描图像的每个像素值如像素值大于K像素值设为255(白色)，值小于等于K像素值设为0(黑色)。不过该方法也是有缺点的，可能导致部分对象像素或者背景像素丢失，这使得二值化结果不能真实反映源图像信息。

方法三：使用直方图方法来寻找二值化阈值，直方图是图像的重要特质，直方图方法选择二值化阈值主要是发现图像的两个最高的峰，然后在阈值取值在两个峰之间的峰谷最低处。

方法四：Otsu（最大类间方差法、大津算法）

matlab中的graythresh即是以该算法为原理执行的。该方法将图像分为前景和背景两部分，背景和目标之间的类间方差越大,说明构成图像的2部分的差别越大,当部分目标错分为背景或部分背景错分为目标都会导致2部分差别变小。因此,使类间方差最大的分割意味着错分概率最小。

算法步骤如下：

设图象包含L个灰度级(0,1…,L-1)，灰度值为i的的象素点数为Ni ，图象总的象素点数为N=N0+N1+...+N(L-1)。灰度值为i的点的概为：

P(i) = N(i)/N.

门限t将整幅图象分为暗区c1和亮区c2两类，则类间方差σ是t的函数：

σ=a1\*a2(u1-u2)^2

式中，aj 为类cj的面积与图象总面积之比，a1 = sum(P(i)) i->t, a2 = 1-a1; uj为类cj的均值，u1 = sum(i\*P(i))/a1 0->t,

u2 = sum(i\*P(i))/a2, t+1->L-1

该法选择最佳门限t^ 使类间方差最大，即：令Δu=u1-u2，σb = max{a1(t)\*a2(t)Δu^2}

参考：（<https://en.wikipedia.org/wiki/Otsu%27s_method>）

补充：均衡直方图二值化？？

# 图像细化

骨架化：通常也将细化描述为“骨架化”、“中轴转换”和“对称轴转换”。

好的细化算法一定要满足**下面几个条件**：

1. 收敛性，保证细化后细线的连通性；
2. 保持原图的基本形状；
3. 减少笔画相交处的畸变；
4. 细化结果是原图像的中心线；
5. 细化的快速性和迭代次数少。

## 两类依据是否使用迭代运算可以分为两类：

1. **非迭代算法**：一次即产生骨架，如基于距离变换的方法，游程编码细化等。
2. **迭代算法**：即重复删除图像边缘满足一定条件的像素，最终得到单像素宽带骨架。

迭代方法依据其检查像素的方法又可以再分成**两类**：

* 串行算法：是否删除像素在每次迭代的执行中是固定顺序的，它不仅取决于前次迭代的结果，也取决于本次迭代中已处理过像素点分布情况。
* 并行算法:像素点删除与否与像素值图像中的顺序无关，仅取决于前次迭代的结果。

常用的迭代算法包括：Hilditch、Pavlidis、Rosenfeld细化算法以及基于索引表查询的细化算法等等。

1. Hilditch算法**使用于二值图像**，该算法是并行串行结合的算法。
2. Pavlidis算法通过并行和串行混合处理来实现，用位运算进行特定模式的匹配，所得的骨架是8连接的（没有理解），用于0－1二值图像。
3. Rosenfeld算法是一种并行细化算法，所得的骨架形态是8－连接的，使用于0－1二值图像。

### Hilditch 细化算法的步骤为：

对图像从左向右从上向下迭代每个像素，是为一个迭代周期。在每个迭代周期中，对于每一个像素p，如果它同时满足6个条件，则标记它。在当前迭代周期结束时，则把所有标记的像素的值设为背景值。如果某次迭代周期中不存在标记点（即满足6个条件的像素），则算法结束。假设背景值为0，前景值为1，则：

6个条件为：

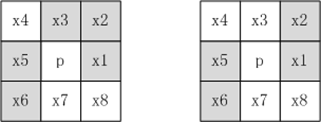
（I）：p 为1，即p不是背景；

（2）：x1,x3,x5,x7不全部为1（否则把p标记删除，图像空心了）；

（3）：x1~x8 中，至少有2个为1（若只有1个为1，则是线段的端点。若没有为1的，则为孤立点）；

（4）：p的8连通联结数为1；

联结数指在像素p的3\*3邻域中，和p连接的图形分量的个数：



4连通联结数计算公式是：

[image](http://images.cnblogs.com/cnblogs_com/xiaotie/WindowsLiveWriter/HilditchC_2CEF/image_12.png)

8连通联结数计算公式是：

[image](http://images.cnblogs.com/cnblogs_com/xiaotie/WindowsLiveWriter/HilditchC_2CEF/image_14.png)

其中，

[image](http://images.cnblogs.com/cnblogs_com/xiaotie/WindowsLiveWriter/HilditchC_2CEF/image_16.png)

（5）假设x3已经标记删除，那么当x3为0时，p的8联通联结数为1；

（6）假设x5已经标记删除，那么当x5为0时，p的8联通联结数为1。

### Burning 算法使用迭代的方法去除图像的边界, 可使用扫描线法来获取边界。设想在同一时刻，将目标的边缘线都点燃，火的前沿以匀速向内部蔓延，当前沿相交时火焰熄灭，火焰熄灭点的结合就是骨架。

### Zhang并行快速细化算法（看网上大多基于此方法）

算法介绍（算法看不懂，老师如果了解的话可以请教下）

Zhang-Suen细化算法通常是一个迭代算法，整个迭代过程分为两步：

1. 第一步：循环所有前景像素点，对符合如下条件的像素点标记为删除：

1. 2 <= N(p1) <=6

2. S(P1) = 1

3. P2 \* P4 \* P6 = 0

4. P4 \* P6 \* P8 = 0

其中N(p1)表示跟P1相邻的8个像素点中，为前景像素点的个数

S(P1)表示从P2 ～ P9 ～ P2像素中出现0～1的累计次数，其中0表示背景，1表示前景

完整的P1 ～P9的像素位置与举例如下：

其中 N(p1) = 4, S(P1) = 3, P2\*P4\*P6=0\*0\*0=0, P4\*P6\*P8=0\*0\*1=0, 不符合条件，无需标记为删除。

1. 第二步：跟第一步很类似，条件1、2完全一致，只是条件3、4稍微不同，满足如下条件的像素P1则标记为删除，条件如下：

1. 2 <= N(p1) <=6

2. S(P1) = 1

3. P2 \* P4 \* P8 = 0

4. P2 \* P6 \* P8 = 0

循环上述两步骤，直到两步中都没有像素被标记为删除为止，输出的结果即为二值图像细化后的骨架。

### 基于索引表的细化方法

**对于边缘点P，它的八连通域是指P点的周围的八个点，我们按如下顺序标识：**

|  |  |  |
| --- | --- | --- |
| **P0** | **P1** | **P2** |
| **P7** | **P** | **P3** |
| **P6** | **P5** | **P4** |

八邻域的数字一共有2^8次方种情况。

首先把256种状态和能不能删除的标记已经对应起来。

从p0开始，顺时针旋转一直到p7组成的二进制数

如：

0 0 0

1 p 1

1 1 1

组成的二进制数就是：0001111，则对应的十进制数就是(二进制转十进制):248。然后找到二进制和对应的删除标记是0还是1。

**对于这八个点的不同取值情况，按如下顺序表示的二进制的值也将有不同的取值**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** | **Sum** | **Delete Enable** |
|  |  |  |  |  |  |  |  |  |  |

**其中Sum的取值从0~255，Delete Enable为1表示可以删除，为0表示不能删除，这样我们就建立了一个索引表deletemark[256]，**

1. deletemark[256] = {
2. 0,0,0,0,0,0,0,1, 0,0,1,1,0,0,1,1,
3. 0,0,0,0,0,0,0,0, 0,0,1,1,1,0,1,1,
4. 0,0,0,0,0,0,0,0, 1,0,0,0,1,0,1,1,
5. 0,0,0,0,0,0,0,0, 1,0,1,1,1,0,1,1,
6. 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
7. 0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,
8. 0,0,0,0,0,0,0,0, 1,0,0,0,1,0,1,1,
9. 1,0,0,0,0,0,0,0, 1,0,1,1,1,0,1,1,
10. 0,0,1,1,0,0,1,1, 0,0,0,1,0,0,1,1,
11. 0,0,0,0,0,0,0,0, 0,0,0,1,0,0,1,1,
12. 1,1,0,1,0,0,0,1, 0,0,0,0,0,0,0,0,
13. 1,1,0,1,0,0,0,1, 1,1,0,0,1,0,0,0,
14. 0,1,1,1,0,0,1,1, 0,0,0,1,0,0,1,1,
15. 0,0,0,0,0,0,0,0, 0,0,0,0,0,1,1,1,
16. 1,1,1,1,0,0,1,1, 1,1,0,0,1,1,0,0,
17. 1,1,1,1,0,0,1,1, 1,1,0,0,1,1,0,0
18. };//索引表

# 轮廓提取

1. 基于边缘的方法

分为：基于边缘检测和基于边缘分组

基于边缘检测：边缘检测算子+去除杂点、冗余边缘、修复边缘（噪声敏感）（基于梯度的方法可以归纳到这里）

基于边缘分组：线逼近算法去除噪声+边缘点逐步组合合并成（噪声敏感）

1. Opencv中提供了很多关于图像轮廓处理的函数

这里我用cvFindContours函数来提取轮廓，首先将图像二值化，因为这个图像必须是一个二值图。得到二值图的方法有很多，这里我采用的是Opencv中的cvThreshold函数，就不需要自己写二值化的相关函数，通过设置阀值来得到相应的二值图。最后用cvDrawContours函数将提取的轮廓画出来。函数cvFindContours的第一个参数就是要进行提取轮廓的目标图像。

问题：并行串行不知道

GDIPlus 是什么