● 担当者必须彻底理解了算法的意思后,再进行实装.

★ 应思考讨论算法的意思及合理性,或不合理性,找出问题,提出问题.

● 必要时可直接联系河边----哪怕有微小的疑问,疑惑,或**不确定性**也要提出来!

★ **非常期望**得到大家的有效的IDEA, 批评性意见,或建议!

为此, 建议各担当者用自己的语言整理一下算法----这样既利于整理思路, 深入细致地理解处理要求, 又利于记忆(备忘), 以及发现问题----甚至最后的DOCUMENT也可用到.

**基本概念及用语解释**

**图像坐标系：** 以图像的最左上角的PIXEL为坐标系原点, X轴的正方向为水平向右, Y轴的正方向为垂直向下.

**WORK (WI)： 图像检测对象之整体**-----如电容, 线圈, 电路板

**REFERENCE图像(RW)： 事先准备好的典型的GOODWORK图像(**NORM**),**或典型的BAD WORK图像----一般无” BAD”限定的情况指的是NORM.

**TEST图像(TW)： TEST WORK图像**

**OBJECT (OI)： WORK的局部图像**----WORK图像上的某一有特殊意义的部分------电容上的两个电极, 电路板中芯片,或管脚,等等.

**TOI：TW上的OI**

**ROI：RW上的OI**

**TEMPLATE坐标集(TXY)： TXY**中的坐标是**RW上的某一局部图像(OBJECT)的**local坐标, 即以包围该TEMPLATE的最小有向矩形的**中心**为坐标系原点坐标

(此点称为TEMPLATE的几何中心,坐标 ＝(0,0) in **TXY** -----**设此中心坐标在RW上的坐标为**(*xc* , *yc* ), 则**TXY = {** (*x*, *y*) **-** (*xc* , *yc* )**}** ).

**TEMPLATE(TI )： 某标准OBJECT的局所正规化图像;其上的所有坐标保存在TXY中.**

**TI**的正规化画像值 ----- 事先算出.

TEMPLATE与TEST图像上的OBJECT之间的匹配要通过**旋转表**来进行.

注意,

各个**TXY**是独立存储的,但各个**TI**是保存在与WORK图像的外接矩形大小相同的FLOAT MATRIX中的相应位置上的-----参照时应利用HARD ADDRESSING 功能.

**回转角：** 在未做特别限定的情况下, 指的是与图像坐标系中的X轴正向的夹角----- 注意, 正角的方向为 **顺时针**

**旋转表：** 即”**旋转的TEMPLATE坐标集**”, 或“**坐标旋转映射表**” , index = ***i*** 的旋转的**TXY**中的坐标为将**TXY**进行*θ* ( = min*+ i\*Δ*)回转后得到的坐标.

min: 最小回转角 (如: －20°); *Δ=* 角度 *resolution* (如: 0.2°)

－min /*Δ*对应原始的(无回转的) **TXY**.

通常旋转范围max = －min; min /*Δ*为整数.

注意,

**1. 旋转表**中的坐标与**TXY**中的坐标是一一对应的,相对应的坐标有相同的memory-index.

**2. 旋转表**中的坐标一般为小数, 对应到TEST图像上时需要对TEST图像插值(INTERPOLATION).

===== (*xc* , *yc* ), **TXY**, 诸旋转的**TXY**, 皆预先算出-----由图像登记及学习APP完成-----暂时与软件学院无关, 但在系统起动时, 这些DATA要从XML文件中读入到GPGPU的MEMORY中 =====

global memory int errmap[H][W] := 0; // ERROR MAP(相当于一种图像), 用来标记TEST图像上对应位置上的PIXEL ERROR. H,W : the dimension of TEST image.

\*============================================================\*

Guided region 坐标集的取得

给出一个坐标集合, 它可能表示一个闭合或非闭合的轮廓,亦或是一个特定形状的区域,记录与这个集合上的任意某点的距离小于指定半径的PIXEL的坐标,则得到一个新的坐标集合.

------此功能的实现与DILATION算法很相似,完全可实现并行化!

在任何算法中,当涉及直方图(HISTOGRAM)时, 若没有特别指出,

都不得用除以总数的方法进行正规化.

也就是说,通常只用整数的HISTOGRAM值.

**基本约定**

1. **IMAGEDATA及存储FORMAT**

IMAGE 为GRAY SCALE, 8bit(unsigned char )

存储FORMAT：左上🡪右下、行順為先,例：

画面表示：

1 2 3

4 5 6

7 8 9

unsigned char\* imgData = {1,2,3,4, 5,6,7,8, 9 }; // 必须是线性连续的MEMORY

**样本画像由OKANO提供.**

1. **画面座標系**

画面左上為座標原点(0,0), X軸の正方向是右；Y軸的正方向是下。

即, X坐标对应列号, Y坐标对应行号, 0-based index.

如无特别要求, 在处理时不对IMAGE边缘进行扩张(图片填充边界), 根据需要适当缩小处理范围即可-----特别是ROI范围较小的图像,一般不用考虑边缘问题.

1. **图像变量的基本构造** -----以下说明基于这个构造

typedef struct imageStruct {

int imgWidth; // 图像宽;

int imgHeight; // 图像高;

int roiX1; // roiX1: ROI的左上角的横坐标;

int roiY1; // roiY1: ROI的左上角的纵坐标;

int roiX2; // roiX2: ROI的右下角的横坐标;

int roiY2; // roiX2: ROI的右下角的纵坐标;

unsigned char\* imgData ; // image data

} Image;

0≦roiX1< roiX2 < width;

0≦roiY1< roiY2 < height;

1. **由小数向整数的转换问题**

没有特别说明时,皆采取四舍五入的原则.

1. **小数坐标值**

坐标值一般为整数,但如果坐标是以FLOAT型定义的, 如果没有特别指定, 则意味着其对应的图像值需要由**HARD** INTERPOLATION来获得.

**HARD** INTERPOLATION ：GPGPU所具有的一种线性INTERPOLATION方式.

1. **快速计算技巧问题**

尽量将小数计算转换成整数计算;

尽量将开方计算转换成平方计算;

将长LOOOOOOOP变成短LOOP.

1. **对只计算一次便可的计算结果,应放在LOOP外,**

**以及其它的可以避免二次计算的地方**

1. **inline的充分利用**
2. **用MEMORY的增加来换取执行速度的提高**
3. **HISTOGRAM,在没有强调的情况下,不要除以总数来正规化.**

**如果该DOCUMENT中出现日语片段, 请不要介意,基本上是与大家无关的内容----个人笔记而已**

Q1.  射影变换(Projective Transformation/ projection transform)是比较有概括性的线性变换, 共有9个参数,可以表现几何形状的边缘平行及不平行的变换,

也可表现几何图形的相似性变换----即各向均匀地扩大缩小.

      由此想得到一个比较限定的变换表达式 :  带有各向均匀地扩大缩小的, 且将矩形转换成梯形(台形)的坐标映射关系式----这相当于一种特化(退化)了的射影变换吧?

        梯形:  有两边是平行的, 而另外两边的倾斜是对称的.

Q2.   包围一个比较复杂的几何形状(点的分布)的最小有向矩形的中心是否与由Hu moment算出的中心一致? 最小有向矩形的中心又是怎么求出的?

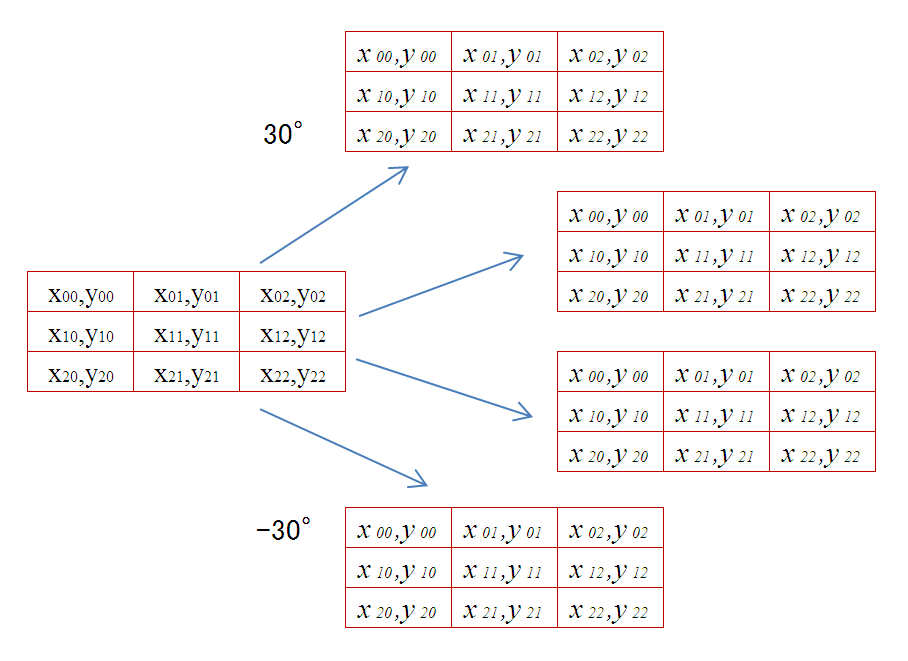
**大量reference WORK image, 以及各种object(a meaning part of WORK image)的坐标set作为既知DATA**

**(已由别的程序预先生成, 一旦读入到GPGPU的内存里,将永远保存).**

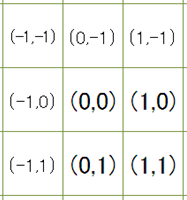
**旋转表実例**

**在GPGPU的MEMORY中应生成一个坐标回转映射表, 例如,**

**下表的其SIZE = 3×3; 回转角范围 = [ -30**°**, 30**°**]; 回转角RESOLUTION = 0.2°**

****

中心坐标为0的相对坐标BLOCK,



**图一**

**因此,共有301个坐标回转映射表被生成到GPGPU的内存中(DATA TYEP : FLOAT).**

**各个坐标回转映射表的名称:**

**XYR[0] : -30回转；XYR[1] : (-30+ 0.2)回转; ….; XYR[i] : (-30+ 0.2\*i) 回转; 　这里的角度单位为度°, i = 0,1,2,…, 300**

**特别是XYR[150]为无回转**

表自身的index

**例如:**

**与-20°对应的index = 100;**

**与 20°对应的index = 250;**

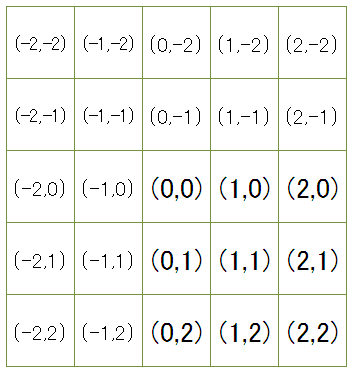
**以后,所有坐标变换都是通过查表的方法进行的.**

**注意, 在存储时可用两个FLOAT型的 MATRIX分别保存变换后的X坐标值和Y坐标值,即XYR 🡪 XR, YR**

简单地说,就是制作若干个表,每一个表表示按某角度回转后的坐标值.

旋转前的表是一个相对的坐标集合----值为连续的2D MATRIX,中心坐标为(0,0).

例如:



将此坐标表中的各个坐标旋转30度,可得另一个坐标表----旋转前后的各组坐标在表中的位置相同.

这样,任何一个图像域(dimension ≦ 上表的dimension),只要其中心与上表的中心合致,便可得到其30度旋转后的相对坐标值.

当然,表中的元素为一对----两个坐标(x,y), 在MEMORY表现上是分成两个变量好呢, 还是结构稍复杂一点的一个变量好呢,

请大家考虑决定-----此表要在GPGPU的MEMORY中生成,要考虑参照此表的效率.

但各个角度的转换表是预先生成好的, 并在系统起动时,从文件中读出复制到GPGPU特定的MEMORY上.

此表的利用法,在<<第一段階開発項目>>有相应的例子.

每次检测开始的第一步是总体匹配----用1/8缩小[注]了的,具有不同回转角的REFERENCE图像(training image) 分别在1/8缩小了的TEST图像上进行匹配(参见后面的"整体相似性指标")----

以1/8缩小了的TEST图像的中心为匹配的摄动中心,在指定的某个范围内进行匹配摄动, 以便找到最大的正规相关值-----由此得到TEST图像的回转角以及原大小的 REFERENCE图像的中心在原大小的TEST图像的上的坐标(X0,Y0)……注意,在本PROJECT中,图像的坐标系是以图像的左上角的PIXEL为原点的.

之后,用各个原大小的TEMPLATE对相应的原大小的OBJECT进行的匹配,都是以(X0,Y0)为参照,确定匹配的摄动中心的.

(而具体地见下面的**“WORK and OBJECT 匹配”**)

[注] 这里的1/8缩小是用概率法生成的缩小(见1/N缩小图像的生成).

**注意**

1. **由于检查对象形状的复杂性, 这里的所有匹配处理都是在指定坐标SET(既知)下进行的,而不是简单地进行两个方形范围间的匹配.**
2. **坐标SET中的坐标值是相对于坐标SET所表示的形状中心的相对坐标-----即形状中心的坐标为(0,0).**

**第一阶段的GPGPU并列化IMPLEMENTATION项目及有关TEST**

1. 坐标变换表的并行生成----在系统初始化时
2. 1/N缩小图像的生成

**三. WORK and OBJECT 匹配**  ( 河边 算法说明, **较大! ----**稍后给出)

四. **经由坐标映射的图像相关计算及局部异常检查** (標準正相関係数的統計and 相関係数map作成) ( 河边 算法说明, **较大!** ----稍后给出)

五． **smoothVector**  (河边 算法说明)

六． Thinning (河边C++ CODE) 【参照 NVIDIA LIB可能】

七． **二値化方法** (河边 算法说明, 参考C++ CODE ) 【参照 NVIDIA LIB可能】

八． Image Enhancement (image *ϒ*变换, log 变换) [理工大软件学院**无关**] (河边 算法说明)

八．Our Salient region detection method (ver1.0) [应作] 　 (河边 算法说明)

九. **get Object Coordinates Set** (or 凹凸多辺形的生成 or 有連結性的**封闭**輪郭**的获得**) (河边 算法说明)

十. **几何形状方向等的測度計算**  (河边 算法说明)

十一. Local clustering (scale level 別) (河边C++ CODE)

十二. Freckle Filter ([斑点除去フィルター](http://ejje.weblio.jp/content/%E6%96%91%E7%82%B9%E9%99%A4%E5%8E%BB%E3%83%95%E3%82%A3%E3%83%AB%E3%82%BF%E3%83%BC)) (河边 算法说明)

十三. **ROBUST EDGE DETECTION ( RED)----对判定边缘欠损有效** (河边 算法说明)

十四. **简易的显著域DETECTION** (河边 算法说明)

十五. Hough変換(直線, 円, 矩形的检出) ( 河边 参考C++ CODE parameter: bin width(pixels)) 【参照 NVIDIA LIB可能】

十六. **5個前回没完成or 没徹底完成的項目**

1. 连通域生成法的交叉确认;
2. 高速有效的Region convex Hull求出;
3. 高速有效的最小有向外接矩形求出[2的基础上];
4. **三种快速SORT法**的交叉确认---速度评价
5. **Affine 变换, by using 河边CODE**

十七. **精确的MAGNITUDE OF GRADIENT 计算法( PMM)**

十八. **ROBUST REGION MATCHING ( RRM)**

**除要对上述项目进行单体DEBUG和TEST**

**↑↑↑↑↑　开发期限3个月　　↑↑↑↑↑**

/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

1. **旋转表**的并行生成----在系统初始化时

由m指定回转角的下限, 由n指定回转角的上限. 角度的分解度为0.2°

m,n可正可负的整数.

前面的例子, -30为回转角的下限, 故m = -150; 30为回转角的上限, 故n = 150

用两个FLOAT型的 MATRIX分别保存变换后的X坐标值和Y坐标值:

XR[0], XR[1], …. , XR[N]

YR[0], YR[1], …. , YR[N]

XR[0], YR[0] 对应回转的下限角; XR[N], YR[N] 对应回转的上限角.

Q1：回旋角的分解度固定为0.2度？还是通过参数设定？

A1: 0.2是最小回转单位(固定),回转都是0.2的整数倍.

Q2：如果分解度不能整除回旋角上下限差时，如何处理？报错？扔掉上（下）限值，还是保留上（下）限值？

A2: 外界指定的回转角度若不是0.2的整数倍,可调整成0.2的整数倍----保证包含外界指定的范围..

Q3：旋转的中心点是坐标原点？还是通过参数设定？旋转后是否需要平移与缩放？

A3: 这个项目仅仅是制做出若干个坐标变换表

比如指定一个HW的四方形,以令其几何中心处的坐标为00, 以这个中心点进行 …, -0.4, -0.2, 0, 0.2,0.4, … 度的旋转, 便可得到若干个坐标旋转变换表T0, T1, T2,….

假如Tk是对应-0.8度的旋转表, 则若要对某指定的坐标(x, y)进行-0.8度的旋转变换时,只要直接查表即可, 即, x’ = Tkx[y][x]; y’ = Tky[y][x]

Q4：每次调用该算法完成一个坐标点的多种旋转角度的计算？还是一组坐标点，坐标点如何给定，通过参数？还是其他的方式？

Q5：所谓“并行生成”是指单个坐标的各个回旋角度计算的并行，还是多个坐标的计算的并行，还是两者皆有？

A4,5: 当然是各个回旋角度间, 及各个回旋角度下的各个坐标间的计算的并行----并行嵌套!

角度的范围在起动时由外界决定的----应该是0.2度的整数倍.

在起动时, 快速地(并行地)算出这个回转变换表,存储到适当的**GPGPU MEMORY**中去,以备以后进行坐标转换时直接查该表得到坐标转换结果.

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

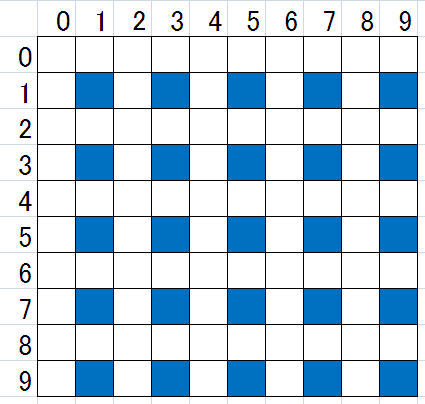
1. **1/N缩小图像的生成**

**解说**

见下图----抽样点之间应有N-1个PIXEL的间隔---即,

每隔N-1个PIXEL採样一次-----用GPGPU来实现此处理时, 缩小后的位置(threadIndex.x, threadIndex.y),

对应原图像的 ( N\*(threadIndex.x + 1) - 1, N\*(threadIndex.y + 1) - 1)上的点.

****

这样各个被抽出的点与其周围的点(没被抽出的点)的图像值将用来确定缩小后的图像的对应点上的图像值.

被抽出的点与其周围的点(没被抽出的点)形成**的DIMENSION为** **(2N-1)\*(2N-1)**

如果被抽出点为边缘点或接近边缘, 其周围的点(没被抽出的点)**的DIMENSION不足为 (2N-1)\*(2N-1),** 则采取NVIDIA提供的图像边缘扩展法进行图像值的统计或计算.

1. **优势(画素値)法**

从原图像抽出x,y皆大于0,且皆为N的整数倍的PIXEL p, 将p的辉度设为该p的**(2N-1)\*(2N-1)** neighbor的前3个HISTOGRAM PEAK加权平均值V.

即, 利用以前开发的并列HISTOGRAM统计CODE求(2N-1)\*(2N-1)图像的HISTOGRAM.

设前3个PEAK对应的辉度值分别g1, g2, g3,对应的画像个数分别为c1,c2,c3.

则V = (g1\*c1+ g2\*c2+ g3\*c3) / (c1+c2+c3) -----取四舍五入取整数

1. **概率(画素値)法**

从原图像(inimg) 中，抽出x,y皆大于0，且皆为N的整数倍的像素点 p，

**随机地**从p所对应的原图像上的(2N-1)\*(2N-1) neighbor中选定一个位置,并以该位置上的PIXEL值作为p在缩小后的图像上的辉度值.

即要在**(2N-1)\*(2N-1)范围内随机确定一个位置……产生一个0到(2N-1)\*(2N-1) -1范围内的随即整数----简单实用的随机算法即可.**

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

这里有个麻烦点,应注意, 每个要被进行mean shift处理的PIXEL都要从**指定坐标集**中选择! -----但对计算要用到的近傍点不必检查是否在**指定坐标集**中----

在此, “灰度”, “辉度” 为同意词

**五．** **smoothVector** (1)~(6)の量的正規化処理は特徴vectorのelementの間公平な値域(バランスの取れた値域)を得られるためである。

1. 求**指定坐标集**所规定的图像范围内的**各PIXEL** n-近旁的輝度中心値CV, 輝度値標準差SD, 最大灰度非共起系数NC. (近旁SIZE n为参数);
2. 对上述**指定坐标集**所规定范围内的CV, SD, NC分别进行排序;

**灰度中心値**float **CV:**

对以现在PIXEL为中心, 上下左右各为n-PIXEL宽度范围内的辉度值进行排序. 即, WINDOW SIZE = (2n+1)×(2n+1)

去掉排序结果两头部分(各1/3)的值,对剩下的1/3部分求**平均**.

----------------------------------------------------------------------------------------

**灰度値標準差**float **SD:**

用上记CV值求WINDOW (2n+1)×(2n+1) 内的灰度値標準差*σ*

-----------------------------------------------------------------------------------------

**最大灰度非共起系数**float **NC:**

分别计算以现在PIXEL为中心的(2n+1)×(2n+1)邻域中8个方向上(0,π/4,π/2, 3π/4,…)的**灰度非共起系数**c1, c2,…, c8;　求{ c1, c2, …, c8} , 选取其中最大的值作为NC.

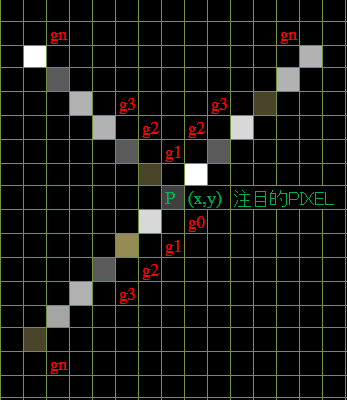
灰度非共起系数计算式:

p : pitch(间距), 外部指定参数.

该方向上的n个pixel的平均灰度值

d : 方向号

对某一方向上的统计计算示意图. (P(x,y): 注目PIXEL)



1. 以各个排序结果的两端5%处的值**分别**作为上述**三种**量的**下限**(最小値)和**上限**(最大値);
2. 分别用**下限**和**上限之间的值**求CV, SD, NC 三种量的**平均值;**
3. 将上述三种量中的大于上限的值置于上限值, 小于下限的值置于下限值;
4. 将各PIXEL的CV, SD, NC减去对应的平均值之后**÷**(**上限**－**下限**),

得到下述中的;

1. have

A: 反复计算过程中的每个yi为那一步的中心点, 最后的这个便是最初的作为出发点的那个PIXEL的”归宿”------收束到了所谓basin中的某点.

这是必然的, 大家不要被那个指数函数所迷惑,那仅仅是个由距中心VECTOR的远近而定的WEIGHT而已!

极端一点看的话,大家可以去掉指数部分(将其看成恒为1)------这样便明显成为VECTOR的算术平均问题!

而算术平均不断进行的结果,必然趋进于出现频度最大的值------无论是SCALAR还是VECTOR.

(*i: index of pixel*)

float α、β：外部指定係数

1. Set the first result as the current input pixel(as a center pixel):

(*x*0,*y*0, CV0, SD0, NC0);

以后,

(*x****i***,*yi*) 現在pixelの座標; (CVi, SDi, NCi) 現在pixelのrange値

1. Compute the next value of the current result— for j = 0 to J-1 compute:

仅用差的norm大小,来评价两个vectorの差, 完全么???

这是一个循环计算过程, 这是此算法的弱点,…..

**Q**： “有无其它替代算法？” ,

**A**: 有些特征空间上的平滑,与时间上的反复平滑几乎是等价的,比如,

De-noising: Bilateral, mean-shift, non-linear diffusion are equivalent[Barash et al., (IVC, 2004)],

有兴趣的同学或老师请研究一下.

.

shift次数iter(i.e., 上述(9)中的处理的回数)为外部指定的参数,与的设定并用-----

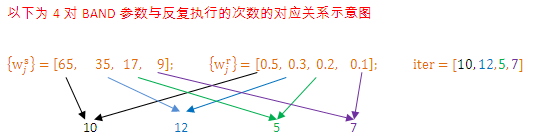
理论上,当所有的() && ( 的标准相关系数接近1) 时,

反复更新计算停止[注] -----但此法无法事先确定循环次数,故用iter, 的选择来决定.

皆为array, 元素个数相同, 以参数的形式给出. 如,

表示在进行(9)中的计算时,

依次用band pair ()进行10次计算;



之后用band pair ()计算12次;

再用()计算5次;

最后用()计算7次.

注: 更严密地说应该是各个之间的各个方位角[方位角的个数＝y的维数]相近,同时, norm也相近时,才可停止.

但由于利用多方位角计算量较大、这里只利用两vector间的夹角(cos计算)做两个vector的相似性测度之一----尽管不是最严格的方法

特征**Vector**的结构体

typedef struct {

int x, y; // pixel的smoothVector循环处理过程中的坐标.

float CV, SD, NC; // pixel的smoothVector循环处理过程中的特征值.

} featureVector

属性matrix的元素=结构体 { // 属性matrix的元素的dimension与作为处理对象的相应的图像REGION的dimension一致.

int x, y; // pixel (x, y)的起始(原始)坐标.

featureVector feature0; // pixel (x, y)的初始(原始)特征值.

featureVector feature; // pixel (x, y)的smoothVector循环处理过程中的特征值.

unsigned char label1; // =lv1 for class 1. 0初始化

unsigned char label2; // =lv2 for class 2. 0初始化

unsigned char class; // =lv1 for class 1; =lv2 for class 2. 0初始化

}

α∈ { 0, 0.01,0.02,0.03,…,0.1, 0.2, 0.3,…, 1, 2,3,…,10, 20,30,…,100 }; // 共38个

β∈ { 0, 0.01,0.02,0.03,…,0.1, 0.2, 0.3,…, 1, 2,3,…,10, 20,30,…,100 }; // 共38个

∈ { 3, 6, 10, 15, 21, 28, 40, 60, 80, 100 }; // space band: 共10个可能VALUE

∈ { 0.1,0.2,0.3,…,1 }; // range band: 共9个可能VALUE

u ∈ { 0.1,0.2,0.3,…,1 }; // 相对weight,共9个可能VALUE

因图像不同而不同, 比如,如果要关掉其它分量的作用, 可将a[lpha](dict://key.0895DFE8DB67F9409DB285590D870EDD/alpha) 或 beta取得很大.

int neighborWidth ∈ { 1,2,….100}; // n

int pitch ∈ { 1,2,3,…100};

### int bandpairNum ∈ { 1,2,3,…10}; // 也就是前次的shiftArraySize

参数INSTANCE的组合数是巨大的,对具体的图象可由LEARNING方法得到最适组合.

Q:“这里有个麻烦点,应注意, 每个要被进行mean shift处理的PIXEL都要从**指定坐标集**中选择! -----但对计算要用到的近傍点不必检查是否在**指定坐标集**中----”。

那么如果n邻域超过图像边缘了怎么办？

A: **smoothVector** METHOD是个对象区域被限定了的 5D的mean shift处理.

这个**指定坐标集**就是或相当于get Object Coordinates Set and Its Contour中的那个”封闭带状B1”.

个人感觉,首先应对指定坐标集中的元素分配GPU THREAD-----让它们一一对应,这样便完成了选择.

另外, 上述各个THREAD执行伊始,应对其所对应的坐标值做LABELING(如200).

即应准备一个 int MATRIX, 其大小由**指定坐标集**中的坐标范围决定,其内各个元素的初始值=0.(以下称这个MATRIX为coordiSetMatrix).

在mean shift处理的循环计算过程中,当结果**Vector**中的坐标成分所对应的coordiSetMatrix上的元素位置处的值=0时, mean shift的结果即为前次的计算结果-----

关与这一点的mean shift处理到此为止----对应的THREAD STOP.

但在计算是要涉及中心坐标近旁的PIXEL所对应的VECTOR,如果这些年近旁的PIXEL的位置超出了**指定坐标集**中的坐标怎么办?

为了简便起见只好用近旁的任何一个非超出的PIXEL所具有的VECTOR来代替了! (对超出实际图像的物理边缘的情况,亦为同样处理)

利用**smoothVector实现二分类(segmentation)的方法**

指定的**smoothVector处理过程完了后,生成了一个**结果**Vector**集合(收束点集合)----**RVS. (**以下总是以RVS中的元素为处理的对象的!**)**

1. 在**RVS**中寻找指定宽度w1内近傍个数最多的**Vector.**

w1 =[w1s, w1r, *hc*], 其元素分别是**Vector**的pixel的坐标bandwidth和图像特征值的bandwidth,方法以及特征vector的方角差阈值.

------- w1s的值一般应取的较大一些 (e.g, 图像的实际处理范围的长和宽的最小者的1/5等).

当**Vector**近傍数相同时, 按外部指定的缩小率对w1r指进行适当的缩小,直到得到一个range bandwidth w1r内近傍个数最多的**Vector .**

将这个**Vector**作为CLASS-1的代表成员(种子成员), 将它的label1(见属性matrix的元素结构体)赋值为lv1.

1. 将由I得到的种子成员的所有近傍**Vector** (指定近傍距离w2=[w2s, w2r, *hc*])**作为**暂定的CLASS-1的成员, 将它们的label1赋值为lv1(暂定成员的传播).
2. 进一步地,对所有label1=lv1的**Vector**之所有近傍**Vector** (指定近傍距离w2=[w2s, w2r, *hc*])**作为**暂定的CLASS-1的成员, 将它们的label1值为赋值为lv1(暂定成员的传播).

此操作循环进行,直到无新的暂定的CLASS-1的成员的出现.

1. 以**RVS**中label1=0 **Vector**为对象, 进行与I相似的操作,生成关于CLASS-2的种子成员,

并确定CLASS-2的种子成员之所有近傍(指定近傍距离w2=[w2s, w2r, *hc*]), 将它们的label2赋值为lv2;

1. 进一步地,对所有label2=lv2的**Vector**之所有近傍**Vector** (指定近傍距离w2)**作为**暂定的CLASS-2的成员, 将它们的label2赋值为lv2(暂定成员的传播).

此操作循环进行,直到无新的暂定的CLASS-2的成员的出现.

1. 以上处理完毕后, 对各个**Vector** , 分别统计其w2**近邻**中的label1=lv1和label2=lv2 **Vector**的累计个数, long sprtCount1, sprtCount2.

if (sprtCount1 > 0 && sprtCount2 == 0) {

class = lv1; // for

}

else

if (sprtCount2 > 0 && sprtCount1 == 0) {

class = lv2; // for

}

else

if (sprtCount1 > 0 | | sprtCount2 > 0) {

分别计算的w2近邻中的关于label1=lv1和label2=lv2的**Vector**的平方差之和:

float distSum1(= Σ for all as the adjacents of and label1=lv1);

float distSum2(= Σ for all as the adjacents of and label2=lv2).

if (sprtCount1 > β\*sprtCount2) class = lv1;

else

if (sprtCount2 > β\*sprtCount2) class = lv2;

else

else class = lv2;

}

float β,α皆为外部指定参数

1. 以上处理完毕后,如果存在class = 0的**Vector X**, 适当增加w值(增量有外部指定), 如果在w范围内, 存在一个class > 0的**Vector Y**, 并且在w范围内与**X**最近**[**注**]**,

则将**Y**的class值赋给**X** 的class. VII操作反复进行,直到不存在class = 0的**Vector.**

**[**注**]** 这里的” 最近的**Vector**” 指的是两**Vector**间的D1最小, 如存在若干个D1相近的**Vector**, 则从中选取具有D2最小关系的**Vector.**

注意

1. unsigned char lv1,lv2的具体数值做为参数由外部指定.
2. 这里所说的**Vector**间的“**近邻**”或“距离”定义为下记的两vector间的差异测度
3. **smoothVector**处理过程收束后,同一CLASS(或SEGMENT)的range差应变得较小;但即使是同一CLASS,收束点的坐标范围也较大.

所以,在计算收束点(∈**RVS**)间的 “距离”时,坐标许容范围ws应定得较大.

两vector间的差异测度:  ~~特徴vector間の差のnorm< w~~~~r~~~~である場合、さらにこの二つ特徴vector間のなす角度(正規相関係数)を調べる必要！~~

设两个特征VECTOR1,2的**smoothVector**结果――收束点(∈**RVS**)的

spatial vector 分别为 ;

range vector 分别为 ;

间的正規相関係数(两vector间的夹角(cos计算))为R12；

当(D1 **<** *hc* && **D2 <**wr &&D3 **<** ws)其中,D1 **=** 1-R12; D2 **=‖‖;** D3 **=‖‖.**

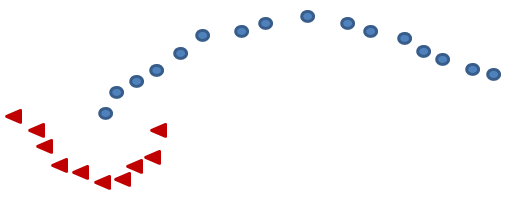
(ws, *hc*, wr皆为外部指定参数) 时, 将这两个VECTOR赋予相同的label値-----

注意, 做为两收束点的VECTOR1,2即便比较接近,但其原始坐标在图像上**未必**呈位置上的**近邻关系!** ----

只是它们的**mean shift**结果相接近----**趋近同一吸盆(basin of attraction)**中的**同一收束点(convergence point)**

在进行segmentation时,把**具有同一class値**的原始坐标所对应的PIXEL, 归为同一SEGMENT.

这一步, 相当于搜索多维空间上的HISTOGRAME 的最大峰值

****

这一步, 相当于搜索多维空间上的HISTOGRAME 的第二大峰值

GPGPU并列化方法探讨------某一特定的vector与RVS中的诸vector间的”距离”(见两vector间的差异测度)同时计算方法

设有如下的matrix RVS[ ][ ] = { {…},{…},{…} };

其中第一index为vector index; 第二index指示一个vector element.

让第一index的值对应一个GPU thread, 即,该thread代表RVS中的一个vector.

于是,对一个特定的vector它与RVS中的各个vector距离可同时算出,结果是否<指定的近傍距离w2也同时判定.

从而可以标识是否是CLASS-1或CLASS-2的成员,或都不是;

也可用此方法寻找特定vector的w2**近邻**vector,并计数.

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

**六．Thinning** (河边C++ CODE) 【参照 NVIDIA LIB可能】

如果PIXEL近旁的2值分布PATTERN 与**指定的PATTERN表**中的某PATTERN相一致, 则

1. 在EDGE图像中消去此PIXEL;
2. 在EDGE坐标SET中消去此PIXEL的坐标.

是A还是B由thinning函数的参数决定.

这里的“删除”指的是被细化掉了的点----如原来很粗的EDGE线上的点,当符合一定PATTERN时将被“删除”,从而完成不断被细化的过程------PATTERN MATCHING法(在此类似于”**Hit**-or-**miss**”)是细化的一种典型方法,如MATLAB中就用过这样的方法.

如果EDGE结果是以IMAGE的形式表现的, 则“删除”意味着PIXEL值被设为低值;

如果EDGE结果是以坐标SET的形式表现的----参见构造体**CoordiSet**, 则“删除”意味着该坐标对,不在结果中存在.

以坐标SET的形式表现的细化结果是经上述细化(“删除”)处理后,剩下来的EDGE坐标的集合.

可以认为是一个2值图像,其内的HEIGHT PIXEL 表示EDGE, 这个EDGE有待细化成1-PIXEL的宽度.

Q1：这里的2值分布PATTERN指的是？PATTERN表指的是否是前文所提及的坐标变换表

A1: 我将给出我的样本CODE(里面有**指定的PATTERN表**),及一篇论文,作为两种IMP的根据.

**请将上述两种IMP结果**与NVIDIA 的LAB的THINING效果相比较----质量和速度的优劣?

Q2：什么是EDGE图像

细化的方法,有我给出样本CODE(里面有**指定的PATTERN表**),及另一篇论文,这可能是提问中所说的，

**第一种方式**和**第二种方式**吧? 都应做一做并与NVIDIA提供的thinning函数进行以下速度及质量的比较.

**质量的比较** 主要指的是----两直线小角度相交时,在交点处的细化结果,是否较好地保持了原来两个直线的形状-----没有出现折线状等.

A2: 可以认为是一个2值图像,其内的HEIGHT PIXEL 表示EDGE, 这个EDGE有待细化成1PIXEL的宽度.

但是,有些情况,EDGE也以坐标集的形式表现.

Q3：Thining的参数是否只有两个备选项，A or B

A3: 1.EDGE值的指定是需要的; 2.当输入为坐标SET时,该坐标SET所参照的图像的指针也是需要的-----参见构造体**CoordiSet**.

Q1:对于需求Thinning算法中的 ，具体的删除方式是什么呢？是直接 删除该点的坐标，其后各点向前平移还是其他的什么方式？

A: 有些细节没有讲出很抱歉!

1. 这一种腐蚀性的细化方法, 即将那个位置的PIXEL值置为0-----假如EDGE值为非0的话.
2. 如果原始EDGE是以坐标集的形式给出的, 为了细化(thinning), 应将其展开于一个2D的BUFFER里---标识成如用1或200等(EDGE值可作为参数由外界指定)表示的EDGE图像.
3. thinning后, 重新收集新的EDGE图像----高值PIXEL的坐标,作为结果坐标SET. 需不需要此步,由外部参数指定.

Q2：对于需求Thinning算法中的 ，该坐标集是以什么样的形式存在呢？需要手动输入嘛？

A: **坐标集**为输入参数的话,由CALL侧生成,其基本形式为本开发的整个PROJECT中通用的形式-----

具体以什么形式为好,请于博士结合GPGPU的特点定一定!

比如,下记MEMBER的构造体如何?

typedef struct {

size\_t n; // 坐标个数

int\* x; // x坐标array

int\* y; // y坐标array

int minX; // x坐标最小值-----用 -1初始化,无须设定此项时一直为-1.

int maxX; // x坐标最大值-----用 -1初始化,无须设定此项时一直为-1.

int minY; // y坐标最小值-----用 -1初始化,无须设定此项时一直为-1.

int maxY; // y坐标最大值-----用 -1初始化,无须设定此项时一直为-1.

Image\* image; // 对象图像

} **CoordiSet**;

or

typedef struct {

size\_t n; // 坐标个数

int\* xy; // (x,y) 坐标array. 这样成为一个MEMORY块,是否更利于GPU MEMORY ACCESS效率?

int minX; // x坐标最小值-----用 -1初始化,无须设定此项时一直为-1.

int maxX; // x坐标最大值-----用 -1初始化,无须设定此项时一直为-1.

int minY; // y坐标最小值-----用 -1初始化,无须设定此项时一直为-1.

int maxY; // y坐标最大值-----用 -1初始化,无须设定此项时一直为-1.

Image\* image; // 对象图像

} **CoordiSet;**

Q3：对于需求Thinning算法中的，这句话具体是什么意思呢？坐标所参照的图像的指针具体指什么呢？

A: 坐标集总是某个图像上的,不可能是架空的,所以,一般应指出那个对应的图像.

图像的指定用图像的指针----即图像的构造体**Image** or **ImageCuda**(到底应是哪个,大家定----前者是对外的,后者是对内的)的指针-----详见<<**CODING RULES (项目编码规范)**>>最终版的开头部分的定义.

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

**七．２値化方法(三种方法)** (河边 算法说明, 参考C++ CODE ) 【参照 NVIDIA LIB可能】

(1) 阈值指定法-----由用户指定一个画像辉度值来二值化

(2) Histogram peak点检查法-----通过检出peak点,来确定适当的二值化乃至多值化阈值

~~A. Histogram peak searching (找出前~~*~~n~~*~~個peak点,以其对应的辉度值为候选的阈值---parameter: bin-width)-----要将此处理独立成一个FUNCTION~~

~~B. 同时(并列地)用不同的阈值对图像进行二值化处理, 用以下方法之一作为确定理想的二值化结果的根据:~~

~~1. 所得的领域是否是两个连通域? 若是, 这两个连通域的面积比最接近预定值者,为理想的二值化结果;~~

~~2. 所得的领域是否是两个连通域? 若是, 这两个连通域的外轮廓的诸HU MOMENT值是否在指定范围内?~~

以1-254内的所有的灰度为阈值,同时生成254个2值化结果(0-1)图像, 然后用以下基准选出一个理想的2值化结果来.

------评价指标有两个.

1. 二者的面积比是否最似于指定值?

## 2值化结果的7个Rotation invariant moments[见后面的Hu Moment计算说明]构成的VECTOR与指定的标准HU MOMENT VECTOR之间的相似度值是否接近+1?

取与指定的指标间的误差皆为最小的2值化结果. 但, HU MOMENT VECTOR之间的相似度值最大者(接近+1)优先. 最终仅选定一个最佳的2值化结果即可.

VECTOR之间的相似度(Pearson's product-moment coefficient):

If we have a series of *n* measurements of *X* and *Y* written as *xi* and *yi* where *i* = 1, 2, ..., *n*, then the *sample correlation coefficient* can be used to estimate the population Pearson correlation *r* between *X* and *Y*. The sample correlation coefficient is written


r_{xy}=\frac{\sum\limits_{i=1}^n (x_i-\bar{x})(y_i-\bar{y})}{(n-1) s_x s_y}
      =\frac{\sum\limits_{i=1}^n (x_i-\bar{x})(y_i-\bar{y})}
            {\sqrt{\sum\limits_{i=1}^n (x_i-\bar{x})^2 \sum\limits_{i=1}^n (y_i-\bar{y})^2}},


用宽度为binWidth的窗口在Histogram上滑动,找到该窗口内包含的Histogram值之和为最大的窗口中心位置, 以其作为候选的辉度阈值.-----这是按旧方法的逻辑来进行的处理!

**但是,求峰值往往费力而无好效果!**

既然我们用并行机构来解决问题, 所以,**没有必要找峰值**,而是以所有的灰度1-254为阈值,同时生成大量的2值化结果(0-1)图像!

然后用以下基准选出一个理想的2值化结果来.

------评价指标有两个.

1. 二者的面积比是否最似于指定值?
2. 2值化结果的7个Rotation invariant moments构成的VECTOR与指定的标准HU MOMENT VECTOR之间的相似度值是否接近+1?

VECTOR之间的相似度=Pearson's product-moment coefficient.

where x and y are the sample [means](http://en.wikipedia.org/wiki/Arithmetic_mean) of *X* and *Y*, and *sx* and *sy* are the [sample standard deviations](http://en.wikipedia.org/wiki/Standard_deviation#With_sample_standard_deviation) of *X* and *Y*.

This can also be written as:


r_{xy}=\frac{\sum x_iy_i-n \bar{x} \bar{y}}{(n-1) s_x s_y}=\frac{n\sum x_iy_i-\sum x_i\sum y_i}
{\sqrt{n\sum x_i^2-(\sum x_i)^2}~\sqrt{n\sum y_i^2-(\sum y_i)^2}}.


If *x* and *y* are results of measurements that contain measurement error, the realistic limits on the correlation coefficient are not −1 to +1 but a smaller range.

(3) 用Histogram searching 法自动找到最佳阈值(最佳二值化结果)

判断的根据:两个领域之间的分散最大(内分散最小)

\begin{displaymath}
\sigma_W^2+\sigma_B^2=\sigma_T^2
\end{displaymath}

$\displaystyle \sigma_W^2$$\textstyle =$$\displaystyle \omega_1 \sigma_1^2 + \omega_2 \sigma_2^2$

$\displaystyle \sigma_B^2$$\textstyle =$$\displaystyle \omega_1\omega_2(\mu_1-\mu_2)^2$

*ω*1*, ω2*は二つのclassのそれぞれの確率

**注意,为了提高计算效率, 不要在求HISTOGRAME时除以总像素数,**

**同样,在其他计算式中也要避免除总像素数!----应按照下面的参考CODE实装并行算法**

## // 判別分析 ＊＊大津法より２値化[Otsu’s Binarization]

// 参照 <<Parallel Implementation of Otsu’s Binarization Approach on GPU.pdf >>

int GetAdaptiveThreshold( IplImage\* image, double\* p\_maxSb )

{

// ヒストグラムを求める(入力は8bit GRAY scale画像)

int hist\_size = 256; // bin size = 1 を意味する

1. 長さLの配列h(i) (i=0,…,L-1)を用意し、全要素を0に初期化する。
2. 对各画像值计数
3. 搜索具有阈值以上的h(i) 所对应的最大画像值Imax和最小画像值Imin

// 長さLの配列h(i) (i=0,…,L-1)を用意し、全要素を0に初期化する。

int *h* [ 256 ]; // 各画像値bin count値(hist value)を取得

for ( int i = 0; i < 256; i++ ) *h* [ i ] = 0; / /全要素用0初期化

// \*\*\*\*在此用*h* [ ]做记数处理----对所有图像里出现的辉度值\*\*\*\*只进行正数的累加计算, 结果放到*h* [ ]的相应位置上,

注意,结果不要除以像素总数

// 判別分析 ましな実装

// 全体の要素数と合計を計算

用*σ*W做separationの尺度和用*σ*B是完全等价的,对一个特定的图像来说,二者之和是固定的常数

*σ*Bを計算するために、個々閾値においての二つのCLASSの平均値a1(k), a2(k) を予め算出する必要

int n = W\*H; // number of all pixel 既知とする。

int u = 0; // C言語において int 即ち “long int”

int **N = n**\*n**;**

int ut = 0;

int nt = 0;

for ( int i = Imin; i < Imax; i++ ) { // 並列可能？[分块]

u += i \* *h* [ i ]; // histogramの総momentを計算

}

// CLASS1の要素数と平均の初期値

int n1 = 0;

long int s1 = 0; // CLASS1の平均の初期値

// 閾値と評価値

int threshold = 0;

float maxEval = 0.0;

float sigmaB;

// ◆◇◆↓は並列可能 １×255 sizeのthread grid

for ( int t = Imin + 1; t < Imax; t++ ) { // t = 255 は必要ない t: **threshold値遍歴**----行き当たりばったり的な閾値確定法

n1 += *h* [ t ]; // 各クラスの要素数と合計を更新/計算

int n2 = **n** - n1; // 残りはもう一つのclassの素子数とする n1/n, n2/n : 二つのclassの現れる確率、即ち公式中の*ω*1*, ω*2

s1 += t \* *h* [ t ]; //

long int s2 = u - s1; // 残りはもう一つのclassのmoment和とする

if (n1 > 0 && n2 > 0 ) {

sigmaB = (float) (s2 –s1)\* (s2 –s1)\*n1\*n2; // *σ*B2の計算法$\displaystyle \sigma_B^2$$\textstyle =$$\displaystyle \omega_1\omega_2(\mu_1-\mu_2)^2$二つclassの確率： *ω*1 = n1/ **n ;** *ω*2 = n2/ **n ;** 二つclassの平均値：*μ*1= s1/n; *μ*2= s2/n 为了计算的效率,结果不要除以像素总数的4次方

} // n1, n2 之中一つでも0値であれば、*σ*B= 0 (sb = 0 )

else eval = 0;

if (sigmaB > maxEval ) { // 閾値・評価値の更新

threshold = t; // 現時点最適閾値

maxEval = sigmaB; // 最大**分離度**値を更新

}

　この関数では、*σ*B2を最大とする $k$を最適閾値とする―――もっとも簡単有効な判拠。

}

如果有指定的面积指标, 对上述得到Otsu’s Binarization 的结果应做阈值的上下调整,以便满足两个二值化部分的面积之比. Q1：（2）中parameter: bin-width指什么

A1: bin-width: 图像值的记数宽度----即,以多大的图像值范围计算(对应)一个HIST.

Q2：（3）中邻域之间的分散最大是指什么？请给出计算公式或参考文献

A2: 参考文献在原文中已给出 <<Parallel Implementation of Otsu’s Binarization Approach on GPU.pdf >> 网上有.

此算法正是为求中邻域之间的最大分散的,同时由此决定了阈值.

Q3：（3）中用Histogram searching 法自动找到最佳阈值是否要遍历所有阈值？或是遍历前n個peak点？

A3: **要遍历有效的图像值**-----请参照给出的CODE.

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

**八．Image Enhancement** [理工大软件学院**无关**]

【*ϒ*变换】

通常,使用上翘的变换曲线, 即 1/*ϒ* = [0.4, 0.8] or *ϒ* = [1.25, 2.5]

**利用预先算好的Look Up Tabel对指定的区域或图像进行----并列化简单**

1. // Gamma補正Look Up Tabel作成
2. **for** (i = 0; i < 256; i++){
3. LUT[i] = (**int**)(pow((**double**)i / 255.0, 1.0 / gamma) \* 255.0);
4. }

由1🡪20得Gamma=2.2; 🡪 2.5

由20🡪1得Gamma=0.46;

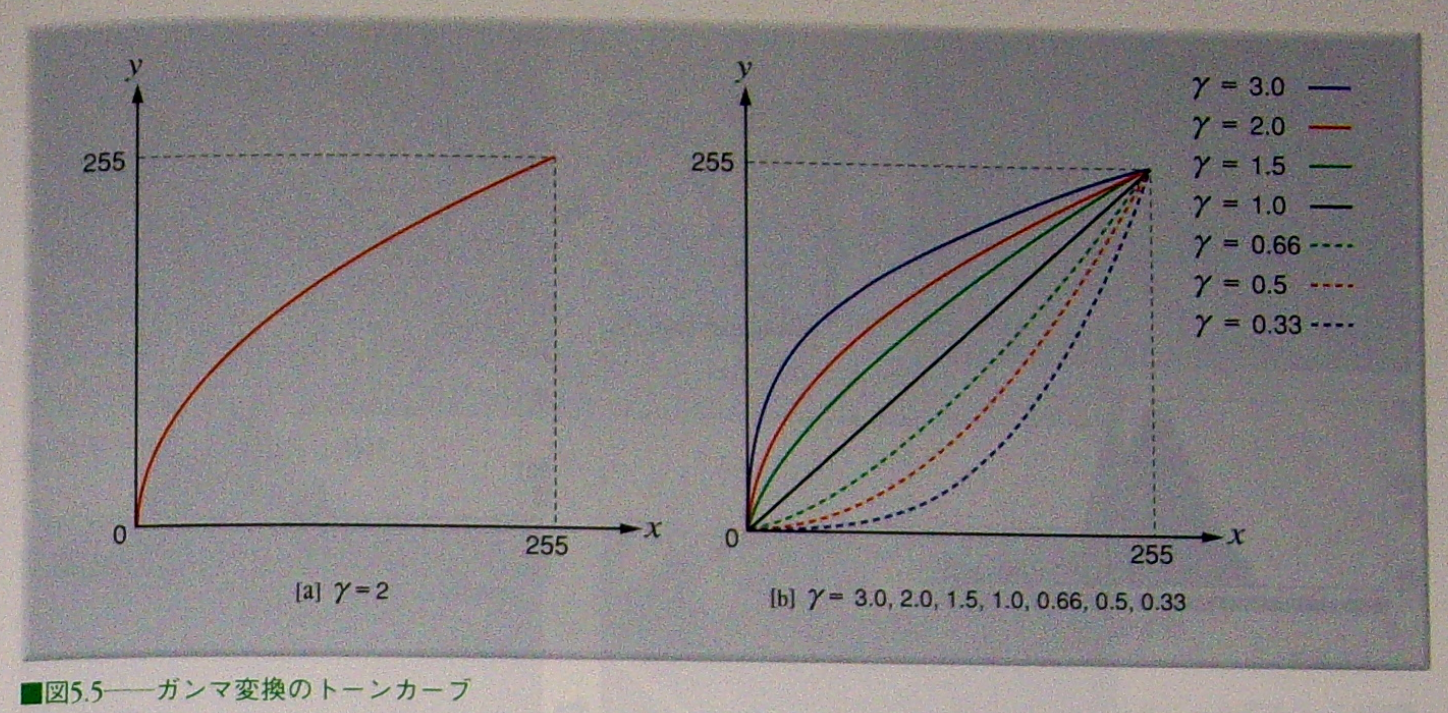
由250🡪200得Gamma=0.08;

可见Gamma=[0.08, 2.5]

这两者间对应的互为倒数

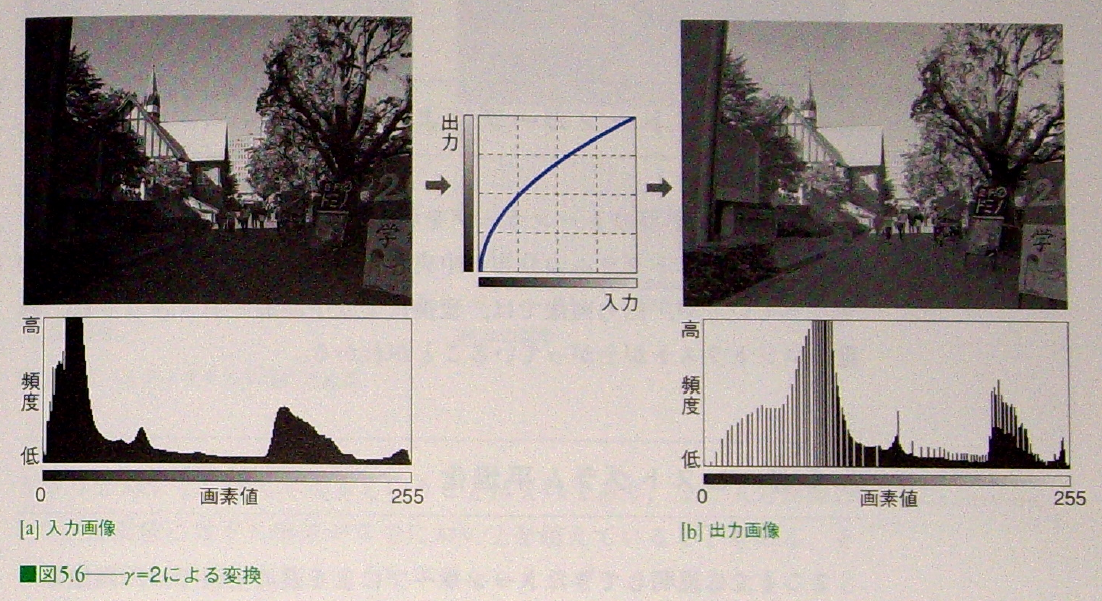
Gamma的可能取值范围,由以下变换范围决定

使1变换成20; 使20变换成1; 使250变换成200



**注意： 是(*x*/255)<1的1/γ, 而不是*x*的1/γ,因此,不要误解曲线的形状.**

上記図[b]中のx := src, y :=dst



================================================:

【log 变换】

**由対数化消除乗性noise** [効果や如何に？]

原画像値g的有効範囲作为**5 ~ 250**

in case 小于5 , 大于250 的值可认为是加性騒音, 直接设定成0和255, 不必进行以下计算.

从g的ln変換結果减去乘性騒音(照明反射騒音) **δ,** 结果(变换后的画像値)记为lg

**lg = ln(g) – δ**

lgの値域：　**L = ln(5) - δ ~ H = ln(250) - δ**

将此线性地映射成0~255范围的值.

于是可有如下总的变换式

G = k\*lg + b

= k\*(lg - L)

= k\*(ln(g) - ln(5))

=

=

=

or

g = [**5,250**]、G的値査表可得

G = 0 if g < 5

G = 255 if g > 250

上述のような対数化処理は、histogram調整[均衡化]に利用しているS-tone curve変換の特例と看做せる。

结果中显然已没有δ,只是査表由g得到G

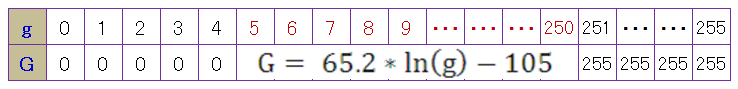
g = [**5,250**]、G的値査表可得

G = 0 if g < 5

G = 255 if g > 250

无并列化IMP的价值----只是对图像值进行査表变换而已!

亦即上述问题归结起来=下表



**八．Our Salient region detection method** [软件学院作]

★★问题点★★

もし、「顕著区」有一定の面積、「顕著区」内のpixelの「顕著値」は非常に小さくなる。

結局、「顕著区」の周辺だけ高い「顕著値」を持つになる----輪郭検出法になる。

このために方法変更を提案：

做下记处理之前, 应用LOCAL CLUSTERING 和Bilateral filter [NVIDIA LIB可用]对图像做适当的平滑处理

1....某pixe的「顕著性値」的求法

(1) 以该pixel为中心,在半径R的范围内, 计算该pixel与其它pixel之间的辉度值差[取绝对值] (if it is color space we will use vector distance) ;

对这些辉度值差进行descending排序,

去掉排序结果中先头的若干值和末尾的若干个值(由hp指定砍掉高位段的hp%, 由lp指定砍掉低位段的lp% ----但一般在高位段砍得少, 而在低位段要砍得很多),

只保留中间部分的排序结果,并对其求平均~~,将这个平均値作为该pixel的「顕著性値」~~----对所有pixel进行这样的计算,形成一个初期saliency map .

改变R值, 重复上述计算,得到若干个初期saliency map

将所有上述初期saliency map进行对应PIXEL上的值累加后,除以所有初期saliency map的个数,

得到(1)的平均saliency map. SM1

注: 诸半径R,作为外部参数以LIST的形式给出. e.g., int\* r = {5, 8, 13,…};

(2) 用σ平滑画像(Gaussian smoothing), 并将平滑结果与经近旁几何平均的图像进行整体差分[取绝对值],得到一个初期saliency map .

改变σ值n次, 重复上述计算,得到n个初期saliency map

将上述n个初期saliency map的对应PIXEL上的值累加后,除以所有初期saliency map的个数,

得到(2)的平均saliency map. SM2

2....对上述(1)和(2)的平均saliency map **加权叠加**,得到一个 "weighted sum saliency map"

SSM = α\*SM1 + β\*SM2 (α , β: 外界指定参数. α + β = 1 )

3.... 对SSM进行连接处理.

连接「顕著性値」相近,位置也相近的PIXEL,得到segmented saliency map ~~----这可由"Mean-Shift "来实现~~【注】

4....对segmented saliency map中的各个REGION 求平均顕著值.

5....按REGION平均顕著值的大小及REGION面积的大小对segmented saliency map 中的REGION进行过滤,筛选,只剩下那些平均「顕著度」 > 阈值, 面积大小符合要求的REGION.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

【注】 参照论文<<Frequency-tuned Salient Region Detection>>

「5.2. Segmentation by adaptive thresholding」

Mean-Shift provides better segmentation boundaries.

We perform mean-shift segmentation in Lab color space.

We use fixed parameters of 7, 10, 20 for sigmaS, sigmaR, and minRegion, respectively, for all the images

(see ”Synergism in Low Level Vision”).

【参照文献】

<<Frequency-tuned Salient Region Detection>> Radhakrishna Achanta

<<Salient Region Detection and Segmentation>> Radhakrishna Achanta

注意, 上面提到的Gaussian平滑中的σ在只是一种象征性的说法, 而这里是直接利用加权系数来实现Gaussian平滑的(不直接用σ计算)。

具体做法是: 指定一个平滑宽度---i.e.,pixel数[奇数]---加权系数的个数,然后计算积和即可. (加权系数之和=1)

当指定平滑宽度时, 加权系数的个数及诸系数的大小便是确定的了, 系数的确定方法如下CODE所示,

可先求出来形成一个表,用平滑宽度来索引.

// n: max-index;

// c: min Gaussian value( 0.2, ....)

double\* makeGaussianWeight(int n, double c)

{

double w = 0;

double sum = 0;

double sc =log(c)/(n\*n); // support span = [log(c), -log(c)]

int n2 = n+n;

double\* weight = new double[n2+2];

for(int i=-n; i<=n; i++) {

double w = exp(sc\*i\*i);

weight[n+i] = w;

sum += w;

}

// 帰一化

for(int i=0; i<=n2; i++) {

weight[i] /= sum;

}

return weight;

}

GAUSSIAN SMOOTHING处理到处都是,所以没有做详细说明,

但还是有需要注意的事项的, 以前给的C CODE只是个例子,

1. 一是说明不直接使用分散量SIGMA,而是用WINDOW的宽度(pixel数),来表示加权平均计算的对象范围;

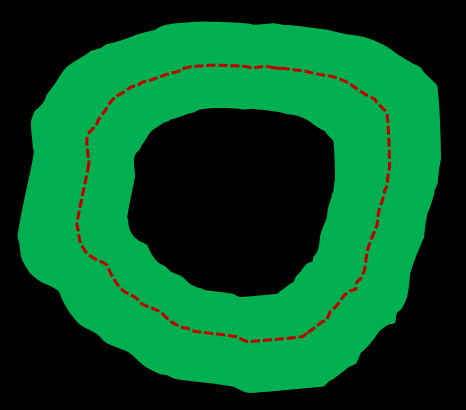
二是说明如何求上述条件下的GAUSSIAN平滑系数-----即, 先指定WINDOW中距中心最远的那个pixel所对应的GAUSSIAN值,然后推出分散系数(or scale)----sc.

1. 那个实装例子是用1D GAUSSIAN SMOOTHING来实现2D GAUSSIAN SMOOTHING的, 虽然效率较好,但需要两个阶段完成,

所以此算法是否对GPU并列化处理有效,应斟酌.

1. 无论如何GPU并列实装, WINDOW内的加权平均计算的权值之和必须为1.
2. 结果的灰度值”四舍五入”
3. GAUSSIAN平滑系数的计算很费时,应预先计算好. 从WINDOW的宽度 = 5pixel数开始, 7,9,11,13----做这5个即可
4. 其它可参照NVIDIA LAB 中的实装.

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/



**九. get Object Coordinates Set** **and Its Contour**(有連結性的封闭輪郭 )

**C1**----上图中的红色点线

B1----上图中的绿色带

B11----中图中的蓝色带(在B12的外侧)

B12----中图中的茶色带(在B11的内侧)

C2----下图中的白色点线

事先要确保一个2D unsigned char **FLAG BUFFER**,

其DIMENSION应包含且稍大于封闭带状B1的范围,

**FLAG BUFFER**初始化值: C1内为**10**; C1外为0

(B12, 及其以内的领域可能包含若干个与B11不同的SEGMENTS)

1. 根据**指定的闭曲线(坐标集)C1**,在**FLAG BUFFER**内生成一个具有指定宽度的封闭带状B1**[注1]**; 将其对应位置设成50[i.e., **FLAG**＝50];



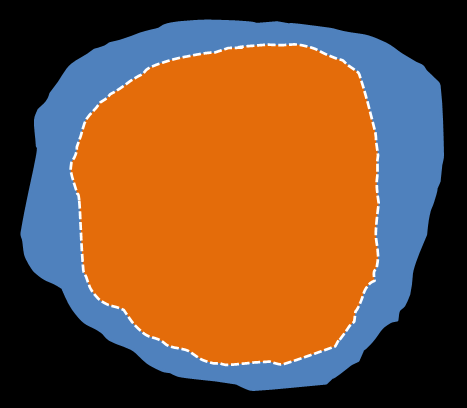
2. 对B1所对应的图象区域进行**smoothVector处理**,得到属性matrix (classification结果);

--------由属性matrix将确定出一个带状的**内外环**B12, B11.  **[注2]**;

3. 从**FLAG BUFFER**的四个边上的点出发[概念上], 由上下左右向**FLAG BUFFER**内侧进行蔓延式搜索B1的最外轮廓点[i.e., **FLAG**＝50].

对搜索经过位置的**FLAG BUFFER**上的值设成100.

蔓延式搜索: 对**FLAG**＝0的点进行**FLAG**值(100)的扩散接力(i.e., 8方向蔓延, 蔓延的条件是相应方向上的邻点位置上的**FLAG**＝0) -----



各个蔓延路径遇到**FLAG** !＝0点时,终止.

当遇到**FLAG**＝50点时,将该点的**FLAG**值改为**150**(外轮廓标识),并记录该点对应的PIXEL所具有的class值(见,smoothVector---属性matrix).

-----最初处理的出发点是所有searching thread所对应的点,它们同时地(**并行地**)进行上述处理----以FLAG值来判定蔓延与否.

开始时各个THREAD之间是平等的,并非预先知道或可指定谁应作为蔓延式搜索的出发点,

故开始时皆为”出发点”, 但searching thread处理的逻辑为

if (对应位置的FLAG == 0) {

8方向蔓延 (蔓延的条件是相应方向上的邻点位置上的**FLAG**＝0);

…….

}

4. 从对上述处理同期, **各跟踪thread全都执行完毕后**,执行以下处理.

5. 从**FLAG**＝**150**所对应的点的位置开始,以**FLAG**＝50的点为对象进行新的蔓延式搜索(搜索过的点的**FLAG**设置成200),

各搜索径路遭遇对应位置的PIXEL具有**没有**登记过的class值为止(即遇到了B11 , B12的交接处),并将与此时相同位置上的**FLAG**值置为250,

increment轮廓点数COUNT.

6. 最后, **FLAG BUFFER**中的**10**值处所对应的坐标集合即为OBJECT的本体,将这些坐标设定于CoordiSet型的变量objxy中.

当函数的参数中存在CoordiSet型的变量contourxy时,

如果无顺序要求,直接将**FLAG BUFFER**中值为250处的坐标记录于contourxy中,作为OBJECT轮廓C2上的点;

如果有顺序要求,则要对**FLAG BUFFER**中值为250的点进行轮廓跟踪处理,最后将有序的轮廓SET保存于contourxy中.

无论哪种情况都要并将COUNT的值设置到contourxy中.

**[注1]**

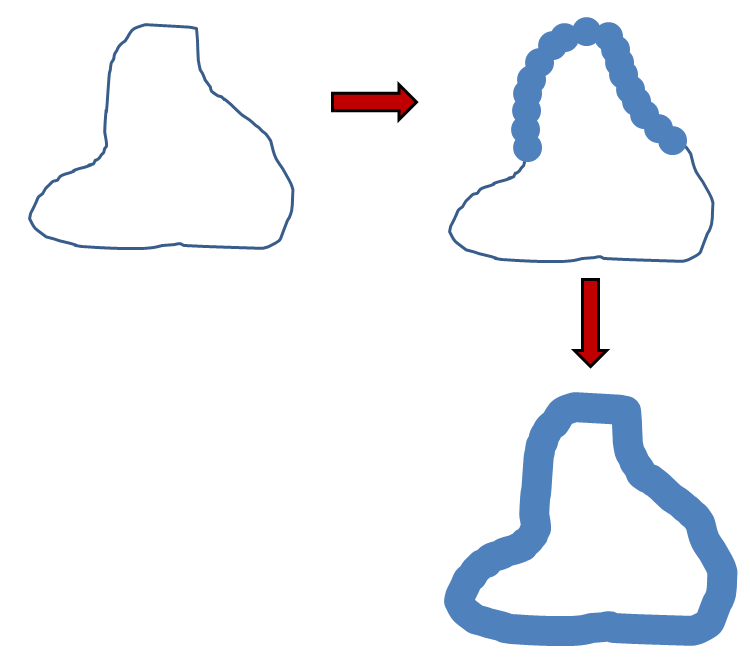
以**C1**上的各点为中心,指定半径的圆的并集,形成了B1.

若(a,b)为曲线C的坐标SET中的元素, (x,y)为其周边的PIXEL,则由满足公式 的PIXEL集合形成了封闭带状B1----

要用这之前的DILATION IMP !

以前已经用GPGPU实装了DILATION算法, 利用此实装便可以实现"封闭带状B1".

效果示意图如下.

封闭带状B1

**[注2]**

这是概念上的,已包含在“利用**smoothVector (mean shift) 实现二分类(segmentation)的方法**”中.

而且这种结果是一种假定, 如果图像不符合此假定, 则用此法不能实现轮廓的检出.

----------APP侧根据需要应该自适应地加大smoothVector的BANDWIDTH,直到B12, B11当中的某一个为单一的LABEL值为止.

其它参见<<**smoothVector**>>中的“利用**smoothVector (mean shift) 实现二分类(segmentation)的方法**”

**[注3]**

按某顺序将两个SEGMENTATION的交界线上的点保存到CoordiSet型的变量中.

(**轮廓线的跟踪法见下面的附录(但自定亦可)**----适合于并行算法的轮廓线的跟踪! )

**注意, 必需从LABELING BUFFER的4边上的点出发!!**

**从4个边同时出发,一方面增加了并行度,另一方面有助于提高FLAG的**”**传染**”**效率-----**

**如对下面的两个形状的区域, 要想同时将所有黑的PIXEL标以相同的FLAG,仅仅进行纵向上的FLAG蔓延**,

**就不如有横向上的FLAG蔓延更易迅速进入到*x*的内部区域----因为*x*的两个开口方向为横向.**

** **

**另外,象以下的图像分布,若同时有横向蔓延的话,也会提高效率.**

****

**当然,上述想法只是从宏观上的可能性的角度上说的,具体地还要看怎么个蔓延法!**

由单纯的几个相同或不同的几何体组成复合形状时,也可利用上述方法, 从包围复合形状的外接矩形的四个边出发,向内蔓延连接点,

各蔓延经络在遭遇几何体的边线点时停止…..直到无新的蔓延点时停止.

----这时外接矩形内,蔓延经络点若与蔓延经络点以外的点相邻则为复合几何体边线上的点.

问题：LABELING BUFFER和FLAG BUFFER的大小是如何确立的？是否要求的是最小外包矩形（如图a所示）？

还是LABELING BUFFER的长为B1最右侧点横坐标与最左侧点横坐标之差，其宽为B1最上方点纵坐标与最下方点纵坐标之差（如图b所示）？

（a） （b）

A4: 作为MEMORY的形状,只能是(b). -----即, MEMORY的长和宽由OBJECT的最大X,Y坐标与最小X,Y坐标的差值直接决定.

**角色分配** ----下记各点皆为N0-pixel

(*x*, y) **start pixel**

(*x*2, *y*2) following出发点(BFSP)

(*x*3, *y*3) following中心点

(*x*4, *y*4) 由following新得到的Border点

(*x*5, *y*5) last 中心点as returning to **start pixel**

**附录 : 闭合轮廓线的跟踪法**

**记,** N0-pixel廓线上的一个点; (*x*, *y*)为轮廓线上的右上方的点; 8N: eight-neighbors

以(*x*, *y*)為8N中心、从 (*x*+1, *y*)的右侧PIXEL出発、按**順時針**捜索N0-pixel；

if (found N0-pixel) {

(*x*2, *y*2) = (*x*5, *y*5) = the first found N0-pixel; // (*x*5, *y*5) 将用于判定是否已**順路返回**.

(*x*3, *y*3) = (*x*, *y*);

}

else return;

loop {

以(*x*3, *y*3)為8N中心、从(*x*2, *y*2)出発, 按**逆時針**捜索N0-pixel (但 (*x*2, *y*2)自身是既定的N0-pixel,不作为捜索对象);

if (found N0-pixel) {

(*x*4, *y*4) = the first found N0-pixel;

}

else return;

当且仅当 (*x*5, *y*5) 成为following中心点,

且由此搜索到的N0-pixel为 (*x*, *y*) 时,说明顺路返回了出发点.

这就是开始时为什么要先顺时针搜索N0-pixel,然后再逆时针搜索N0-pixel的原因.

if ( (*x*4, *y*4) != (*x*, *y*) | | (*x*3, *y*3) != (*x*5, *y*5) ) { // coming back to the starting point.

(*x*2, *y*2) = (*x*3, *y*3) ;

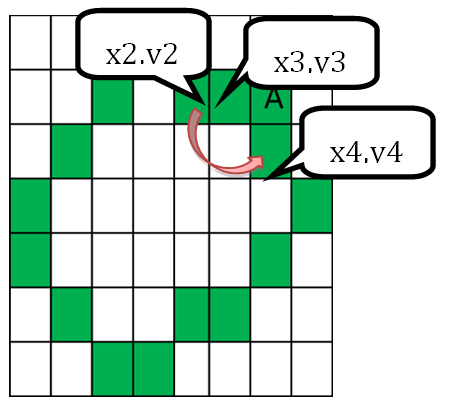
(*x*3, *y*3) = (*x*4, *y*4) ;

}

else return;

}

问题: 用此算法来处理如下的绿色轮廓，将不会经过轮廓上的A点。



-----A点实质上是个傍路(它与其近傍的两点又形成了一个区域----概念上的), 这种曲线跟踪法只取两个傍路之一, 所以A被忽略掉了.

这样并不会破坏轮廓的连续性,一般对由其包围的领域的形状的改变量也很小.

但是, 如果A点恰为矩形的顶点----由于这是一个很精确的顶点, 失去的话确实有些可惜.

增加一些判定或辅助处理可能避免这种现象的发生….由于此曲线跟踪法并行性较弱,增加太多的辅助处理将造成很大的处理时间的浪费.

所以, 很希望用并行性强的跟踪法(以前遇到过,请大家查一查), 有了并行算法后,那么,在处理上的自由度便可大幅提高!

(多曲线之间并行跟踪还是计较容易的, 而对单曲线同时跟踪则需要有巧妙的算法)



对实装九而言, Border Following 处理是在**FLAG BUFFER**上进行的-----以**FLAG BUFFER**上元素值为250处为轮廓点,即N0-pixel.

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

以下, #E\_EROUGH, #E\_ EDEFORM, #E\_ EHUMOM,…为最后需统一编号的ERROR 标识值(int)

**十． 几何形状方向等的測度計算**

(1) **Edge的平滑度計算**-----对给定的EDGE坐标SET(无序)求其所形成的曲线的光滑度

A. 对检查出来的EDGE点进行DILATION处理(并列), 之后THINNING(并列) 并**去除孤立点,** 得到TEST图像的EDGE (**TE**);

B. 对上述THINNING结果**按指定尺度**进行Gaussian平滑[即EDGE上的相邻点的坐标之间进行适当的local加权平均, 求加权平均时的相邻点的个数可指定];

C. 计算**平均误差度**: s = 1**/**(1+r); r = 各平滑前的EDGE点到平滑后的EDGE点的距离的平方和**/**被平滑的EDGE点总数;

D. **平均误差度 > 指定的阈值 →** errmap[.][.] := #E\_EROUGH; // 在ERROR MAP上的3×3的范围内,若存在3个以上的值 = E\_ROUGH的PIXEL,则记录这个3×3的范围的中心坐标----作为边缘故障点;

问题1：Edge的平滑度计算

edge的平滑度计算的输出是平均误差度吗？平均误差度就是平滑度吗？按照算法描述r表示所有点的平滑前后最小距离和，再除以点数，那么整个形状应该只有一个平滑度值吧，

那么errmap[][]=#E\_EROUGH是什么意思，是对哪个边缘点的状态进行赋值吗？

A:　这的确是理解了的提问! 问得很好! 确实,这种情况只应返回一个结果;

但为了标明整个曲线不合格,故只能对errmap的与原曲线的各个坐标点相对应的位置上**皆设定**一个error number E\_EROUGH.

(2) **Edge的异常点检查1----**直接对比法(严格TEST)

**A.** 找出与**TE**坐标SET相关系数最大的GOOD REFERENCE 图像的对应EDGE(THINNING后的) (**RE**);

B. 对**TE**上的点找出与**RE**上的最小距离(mind)点,

if (mind > **指定的阈值**) **→** errmap[.][.] := #E\_ EDEFORM; // 在ERROR MAP上的3×3的范围内,若存在3个以上的值 = E\_ROUGH的PIXEL,则记录这个3×3的范围的中心坐标----作为边缘故障点;

(3) **Edge的异常片段检查2----**MOMENT法(TEST with relaxation) (central moments of order up to 3 ----十.(7)处实现)

**A.** 假定**RE**上各点处的7个HU-MOMENT已预先算出. 计算**TE**上各PIXEL处的7个HU-MOMENT----构成一个vector MV7;

B. 计算上述二者间对应位置上的PIXEL**[注]**的MV7的标准相关系数(cos). **[注] “**二者间对应位置**”** 即, 寻找**RE**上的与**TE**的点最小距离(mind)点.

if (mind < 指定的范围(maxd)) {

进行cos的计算; // maxd 作为参数由外界指定

**HU MOMENT对点的分布往往有较大的敏感性---特别是位置乖离性大的点,**

**所以, 在对曲线使用HU MOMENT之前, 应对其周围充分去噪的,要保证具有连续性,线的粗细= 1 pixel.**

**这里假定上述处理已由别的函数完成.**

if (cos< mincor) // mincor 作为参数由外界指定

else errmap[.][.] := #E\_ EHUMOM;

}

else errmap[.][.] := #E\_ EDEFORM;

曲线的LOCAL MOMENTS计算涉及到对曲线进行双向跟踪问题.即,对当前注目点按两个方向各找出一段连接的曲线段,两曲线段的总长(pixel个数)由外界指定.

然后对由两曲线段构成的坐标集进行central moments的计算,作为以当前注目点为中心的曲线段的moments.

这里所谓的“曲线的双向跟踪”指的是下记算法

假如要计算曲线上的P点处的LOCAL MOMENTS,则从P点的某一近傍点出发,分别进行顺时针和逆时针的曲线following process.

顺逆时针片侧的following长度由外部参数指定. following总长度 = 由顺逆时针检出的曲线上的PIXEL个数之和.

由以上步骤检出的曲线上的PIXEL的坐标,将用于P点处的LOCAL MOMENTS计算.

(4) **直线的线性度计算**

A. 对**TE**的坐标SET~~用最小2乘法求取FITTING直线[~~对NOISE太敏感~~]~~; 🡪 用并行的HOUGH 变换求出直线参数

B. 直线的线性度计算: *l* = 1**/**(1+r); r = **TE**上的各点到FITTING直线的最小距离(垂直距离)之和**/TE**上的点数;

if (*l* < 指定的范围(min*l*)) errmap[.][.] := #E\_ ELINEER;

(5) **圆度的测度 (**RIGION ROUNDNESS )

A. [StructureFactorAnisometryStructureFactorStructureFactorstructureFactor](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#StructureFactor) = 分布的最小有向外接矩形的**短径**/**长径**

B. [StructureFactor[BulkinessBulkinessBulkinessBulkinessbulkiness](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#Bulkiness)](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#StructureFactor) = 分布的最小有向外接矩形面积S/ 分布的实际面积; S = 坐标分布的共分散MATRIX的两个eigenvalue之积のroot =

C. [StructureFactorStructureFactorStructureFactorStructureFactorstructureFactor](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#StructureFactor) = [StructureFactorAnisometryStructureFactorStructureFactorstructureFactor](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#StructureFactor)\*[StructureFactor[BulkinessBulkinessBulkinessBulkinessbulkiness](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#Bulkiness)](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#StructureFactor) = (分布的最小有向外接矩形的**长径**)2 /分布的实际面积 ----円形度の一種?　 eccentricity

D. circularity　= 领域的实际面积 / [PI\*(领域边缘到**领域的几何中心**的平均距离)2]　 // 丸み= 領域の真円度. 円の形状係数 C は 1 です。

// 領域が長いかまたは穴をもつならば、形状係数 C は 1 未満となります。circularitycircularitycircularityCircularityCircularity は、大きいふくらみ、ホールや連結していない領域の特徴を捉える場合に特に意味をもちます。

E. 多角形の辺数的估计法

**可用如下几个指标刻画轮廓的局部特征:**  注意,为简洁起见,各个指标的定义与原始的有出入.

**### 并行地同时求出 ###**

1.指定范围内的**曲线上的点数*S*** (*S*: 对领域来说是实际面积,对轮廓线来说,即,曲线上的点数)

2.[StructureFactorAnisometryStructureFactorStructureFactorstructureFactor](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#StructureFactor)1

3.[StructureFactorAnisometryStructureFactorStructureFactorstructureFactor](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#StructureFactor)2

4.[BulkinessBulkinessBulkinessBulkinessbulkiness](http://www.halcon.de/halcon/download/documentation/reference_jp-10/eccentricity.html#Bulkiness)

5.Circularity

6. Orientation if 付号不一致, 则两局部曲线完全不匹配

7.

8.

9.

10.

p : 面積的中心; -----本质上应用几何中心,但较难,暂且用Hu moment算出的**分布重心**代替

pi: 领域(或分布)上的有效点(对象PIXEL);

a: 领域(或分布)的面積。

计算:

平均距离 [*DistanceDistanceDistanceDistancedistance*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Distance) = sum(||p - pi||) / a

方差距离 [*SigmaSigmaSigmaSigmasigma*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Sigma)^2 = sum(||p - pi|| - [*DistanceDistanceDistanceDistancedistance*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Distance))^2 / a

圆度 [*RoundnessRoundnessRoundnessRoundnessroundness*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Roundness) = 1 - [*SigmaSigmaSigmaSigmasigma*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Sigma) / [*DistanceDistanceDistanceDistancedistance*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Distance) // 圆---相当于边数无限多的多边形

(規則的な)多角形的辺数 [*SidesSidesSidesSidessides*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Sides) = 1.4111 \* ([*DistanceDistanceDistanceDistancedistance*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Distance)/[*SigmaSigmaSigmaSigmasigma*](http://www.halcon.de/halcon/download/documentation/reference_jp-10/roundness.html#Sigma))^0.4724 // if 领域为圆,由此可估计圆的周长?

F. compactness

(区域的轮廓长)**2**/ (4π\*区域的面积)

(轮廓长由THINING化了的EDGE长度来计算----并列性很强----求法参照附录)

(6) 凸面度(convexity)---- **●**判定是否是规则的几何形状的测度**●**

convexity C = ao / ac

ac : 領域的凸包面积[用前次完成的求凸包出程序完成----由多边形顶点求面积有很简单的公式];

ao : 領域自身的真实面积

領域为凸时 (如, 長方形、圆 的形状係数 C= 1 ); 而存在凹处或内部含有孔洞的领域C < 1 .

## (7) 指定座標set的Hu moment (只用Central moments)計算---包含分布的重心和分布的方向的求出

## (参照如下----文献源: <http://en.wikipedia.org/wiki/Image_moment>)

For a 2D continuous function *f* (*x*,*y*) the [moment](http://en.wikipedia.org/wiki/Moment_(mathematics)) (sometimes called "raw moment") of order (*p* + *q*) is defined as

 M_{pq}=\int\limits_{-\infty}^{\infty} \int\limits_{-\infty}^{\infty} x^py^qf(x,y) \,dx\, dy

for *p*,*q* = 0,1,2,... Adapting this to scalar (greyscale) image with pixel intensities *I*(*x*,*y*), raw image moments *Mij* are calculated by

M_{ij} = \sum_x \sum_y x^i y^j I(x,y)\,\!

用此计算*Mij* ; 但要两个版本的, 一是带***I***(*x*,*y*)的, 另一个是***I***(*x*,*y*)≡1时的情况

In some cases, this may be calculated by considering the image as a [probability density function](http://en.wikipedia.org/wiki/Probability_density_function), *i.e.*, by dividing the above by

\sum_x \sum_y I(x,y) \,\!

A uniqueness theorem (Papoulis [1991]) states that if *f* (*x*,*y*) is piecewise continuous and has nonzero values only in a finite part of the *xy* plane, moments of all orders exist, and the moment sequence (*Mpq*) is uniquely determined by *f* (*x*,*y*). Conversely, (*Mpq*) uniquely determines *f* (*x*,*y*). In practice, the image is summarized with functions of a few lower order moments.

Simple image properties derived *via* raw moments include:

* Area (for binary images) or sum of grey level (for greytone images): *M*00
* Centroid: {*x*, *y* } = {*M*10/*M*00, *M*01/*M*00 }

## Central moments

[Central moments](http://en.wikipedia.org/wiki/Moment_about_the_mean) are defined as

 \mu_{pq} = \int\limits_{-\infty}^{\infty} \int\limits_{-\infty}^{\infty} (x - \bar{x})^p(y - \bar{y})^q f(x,y) \, dx \, dy 

where \bar{x}=\frac{M_{10}}{M_{00}} and \bar{y}=\frac{M_{01}}{M_{00}} are the components of the [centroid](http://en.wikipedia.org/wiki/Centroid).

If *ƒ*(*x*, *y*) is a digital image, then the previous equation becomes

\mu_{pq} = \sum_{x} \sum_{y} (x - \bar{x})^p(y - \bar{y})^q f(x,y)

The central moments of order up to 3 are:

\mu_{00} = M_{00},\,\!

\mu_{01} = 0,\,\!

\mu_{10} = 0,\,\!

\mu_{11} = M_{11} - \bar{x} M_{01} = M_{11} - \bar{y} M_{10},

\mu_{20} = M_{20} - \bar{x} M_{10}, 

用前面的*Mij*计算这里的后7个量

\mu_{02} = M_{02} - \bar{y} M_{01}, 

\mu_{21} = M_{21} - 2 \bar{x} M_{11} - \bar{y} M_{20} + 2 \bar{x}^2 M_{01}, 

\mu_{12} = M_{12} - 2 \bar{y} M_{11} - \bar{x} M_{02} + 2 \bar{y}^2 M_{10}, 

\mu_{30} = M_{30} - 3 \bar{x} M_{20} + 2 \bar{x}^2 M_{10}, 

\mu_{03} = M_{03} - 3 \bar{y} M_{02} + 2 \bar{y}^2 M_{01}. 

It can be shown that:

\mu_{pq} = \sum_{m}^p \sum_{n}^q {p\choose m} {q\choose n}(-\bar{x})^{(p-m)}(-\bar{y})^{(q-n)}  M_{mn}

Central moments are [translational invariant](http://en.wikipedia.org/wiki/Translational_invariance).

Information about image orientation can be derived by first using the second order central moments to construct a [covariance matrix](http://en.wikipedia.org/wiki/Covariance_matrix).

\mu'_{20} = \mu_{20} / \mu_{00} = M_{20}/M_{00} - \bar{x}^2

\mu'_{02} = \mu_{02} / \mu_{00} = M_{02}/M_{00} - \bar{y}^2

\mu'_{11} = \mu_{11} / \mu_{00} = M_{11}/M_{00} - \bar{x}\bar{y}

The [covariance matrix](http://en.wikipedia.org/wiki/Covariance_matrix) of the image I(x,y) is now

\operatorname{cov}[I(x,y)] = \begin{bmatrix} \mu'_{20}  & \mu'_{11} \\ \mu'_{11} & \mu'_{02} \end{bmatrix}.

The [eigenvectors](http://en.wikipedia.org/wiki/Eigenvector) of this matrix correspond to the major and minor axes of the image intensity, so the **orientation** can thus be extracted from the angle of the eigenvector associated with the largest eigenvalue. It can be shown that this angle Θ is given by the following formula:

\Theta = \frac{1}{2} \arctan \left( \frac{2\mu'_{11}}{\mu'_{20} - \mu'_{02}} \right)

The above formula holds as long as:

\mu'_{11} \ne 0

The [eigenvalues](http://en.wikipedia.org/wiki/Eigenvalue) of the covariance matrix can easily be shown to be

 \lambda_i = \frac{\mu'_{20} + \mu'_{02}}{2}  \pm \frac{\sqrt{4{\mu'}_{11}^2 + ({\mu'}_{20}-{\mu'}_{02})^2  }}{2}, 

and are proportional to the squared length of the eigenvector axes. i.e., : 点の分布範囲の長; : 点の分布範囲の幅

The relative difference in magnitude of the eigenvalues are thus an indication of the eccentricity of the image, or how elongated it is.

The [eccentricity](http://en.wikipedia.org/wiki/Eccentricity_(mathematics)) is

 \sqrt{1 - \frac{\lambda_2}{\lambda_1}}. 

: 点の分布範囲の長

: 点の分布範囲の幅

**注：すべての対称行列は実数の**eigen value**を持つ[**コーシー**]**

この事実は、[1855年](http://ja.wikipedia.org/wiki/1855%E5%B9%B4)に[エルミート](http://ja.wikipedia.org/wiki/%E3%82%B7%E3%83%A3%E3%83%AB%E3%83%AB%E3%83%BB%E3%82%A8%E3%83%AB%E3%83%9F%E3%83%BC%E3%83%88)によって、今日[エルミート行列](http://ja.wikipedia.org/wiki/%E3%82%A8%E3%83%AB%E3%83%9F%E3%83%BC%E3%83%88%E8%A1%8C%E5%88%97)と呼ばれる概念に対して拡張された

さらに、[直交行列](http://ja.wikipedia.org/wiki/%E7%9B%B4%E4%BA%A4%E8%A1%8C%E5%88%97)のeigen value全てが[単位円](http://ja.wikipedia.org/wiki/%E5%8D%98%E4%BD%8D%E5%86%86)上に分布する[なので、eigen valueは実数に限らない]

: 分布の**扁度**

**扁度**のもう一つの定義：

*S* : 分布域の面積

## Scale invariant moments

Moments *ηi j* where *i* + *j* ≥ 2 can be constructed to be [invariant](http://en.wikipedia.org/wiki/Invariant_(mathematics)) to both [translation](http://en.wikipedia.org/wiki/Translation_(geometry)) and changes in [scale](http://en.wikipedia.org/wiki/Scale_(ratio)) by dividing the corresponding central moment by the properly scaled (00)th moment, using the following formula.

\eta_{ij} = \frac{\mu_{ij}} 
                        {\mu_{00}^{\left(1 + \frac{i+j}{2}\right)}}\,\!

## Rotation invariant moments

It is possible to calculate moments which are [invariant](http://en.wikipedia.org/wiki/Invariant_(mathematics)) under [translation](http://en.wikipedia.org/wiki/Translation_(geometry)), changes in [scale](http://en.wikipedia.org/wiki/Scale_(ratio)), and also [*rotation*](http://en.wikipedia.org/wiki/Rotation). Most frequently used are the Hu set of invariant moments [[1]](http://en.wikipedia.org/wiki/Image_moments#cite_note-.E2.80.9Chu-0):


 \begin{align}
   I_1 =\ & \eta_{20} + \eta_{02} \\
   I_2 =\ & (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2 \\
   I_3 =\ & (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
   I_4 =\ & (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
   I_5 =\ & (\eta_{30} - 3\eta_{12}) (\eta_{30} + \eta_{12})[ (\eta_{30} + \eta_{12})^2 - 3 (\eta_{21} + \eta_{03})^2] + \\
        \ & (3\eta_{21} - \eta_{03}) (\eta_{21} + \eta_{03})[ 3(\eta_{30} + \eta_{12})^2 -  (\eta_{21} + \eta_{03})^2] \\
   I_6 =\ & (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
   I_7 =\ & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] - \\
        \ & (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2].
 \end{align}


The first one, *I*1, is analogous to the [moment of inertia](http://en.wikipedia.org/wiki/Moment_of_inertia) around the image's centroid, where the pixels' intensities are analogous to physical density. The last one, *I*7, is skew invariant, which enables it to distinguish mirror images of otherwise identical images.

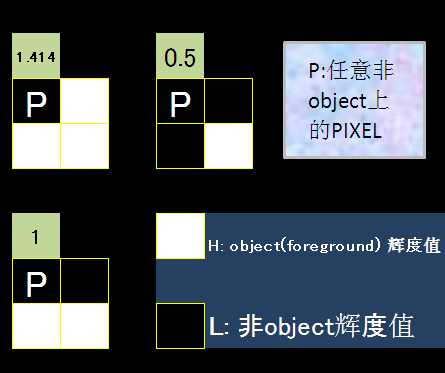
A general theory on deriving complete and independent sets of rotation invariant moments was proposed by J. Flusser[[2]](http://en.wikipedia.org/wiki/Image_moments#cite_note-Flusser-1) and T. Suk.[[3]](http://en.wikipedia.org/wiki/Image_moments#cite_note-Suk-2) They showed that the traditional Hu's invariant set is not independent nor complete. *I*2 and *I*3 are not very useful for pattern recognition, as they are dependent. On the original Hu's set there is a missing third order independent moment invariant:


 \begin{align}
I_8 =\ & \eta_{11}[ ( \eta_{30} + \eta_{12})^2 - (\eta_{03} + \eta_{21})^2  ] - (\eta_{20}-\eta_{02}) (\eta_{30}+\eta_{12}) (\eta_{03}+\eta_{21})
 \end{align}


**注:** **领域的几何中心** or **分布的几何中心**, 指的是该领域分布的最小外接有向矩形的**几何中心-----应注意与分布重心的区别!**.

**附录**

轮廓长的并行计算



分别统计上述PATTERN在指定图像范围内的个数----P的位置为任意,不限于左上角.

PATTERN的种类有3种

1. P的3-近邻内有1个OBJECT上的PIXEL;
2. P的3-近邻内有2个OBJECT上的PIXEL;
3. P的3-近邻内有3个OBJECT上的PIXEL;

记上述COUNTER分别为A,B,C, 则OBJECT的轮廓长度

L = 0.5A + B + 1.414C when C > 0

L = A + B 　 when C = 0

Q1：#E\_EROUGH, #E\_ EDEFORM, #E\_ EHUMOM…，这些宏定义的意义能否说明一下？

A1. 抱歉! 这些个ERROR NUMBER,最后要用汇总以下----以现在也可通过enum定义一下, 尽管各个值可能不是最终的.

Q2：errmap数组的大小是多少，是图像大小还是边界像素个数的大小？初始值是多少？

A2. errmap的DIMENSION完全同TEST图像.

Q3：Edge的异常点检查2—moment法：点最小距离采用何种度量公式？

A3. << **“**二者间对应位置**”** 即, 寻找**RE**上的与**TE**的点最小距离(mind)点 >> 用两个PIXEL的坐标求Euclidean distance即可.

Q4： 指定坐标set的Hu moment，只用Central moments計算？那么后面的Hu moment（I1，I2，…,I7）是否就没有计算的必要了？

A4. YES, 没有计算的必要-----作为参考资料一起COPY过去的,反而添了麻烦!

问题1：Edge的平滑度计算

edge的平滑度计算的输出是平均误差度吗？平均误差度就是平滑度吗？按照算法描述r表示所有点的平滑前后最小距离和，再除以点数，

那么整个形状应该只有一个平滑度值吧，那么errmap[][]=#E\_EROUGH是什么意思，是对哪个边缘点的状态进行赋值吗？

A:　这的确是理解了的提问! 问得很好! 确实,这种情况只应返回一个结果;

但为了标明整个曲线不合格,故只能对errmap的与原曲线的各个坐标点相对应的位置上皆设定一个error number E\_EROUGH.

另外，后面的 “在ERROR MAP上的3×3的范围内,若存在3个以上的值 = E\_ROUGH的PIXEL,则记录这个3×3的范围的中心坐标----作为边缘故障点;”是边缘故障点检测，与平滑度计算有关吗？

A:　这是最后由其它FUNCTION进行的ERROR判定处理时的根据,与这里无直接关系.

问题2：Edge的异常点检查2——MOMENT法

这里面的Hu-MOMENT也需要并行实现，而不是采用central moment？

A:　请见新版的说明.

问题3：圆度的测度

该算法的目的是计算从A到F六个形状属性值吗？这些属性和圆度有什么关系？

A:　是的! 它们从不同的角度反应了"圆度"------如,在各个方向上的扩张的对称性,周长与面积的比等.而且,所谓的"圆"亦可看成是凸多边形的一种极限情况.

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

十二.  **Freckle Filter**

src : 原图像

dis: 处理后的图像BUFFER ( 0初始化 )---应该保存在GPGPU SHARED MEMORY中?

**方法1:**

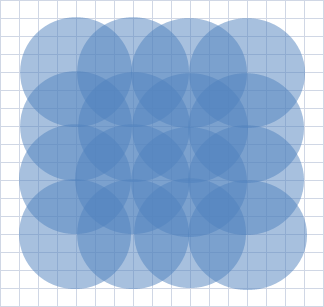
分别以src中的各个PIXEL为中心, 计算指定半径的**圆周上[注1]**的PIXEL的辉度值的方差(long variance),

if (此方差<指定的值(long varianceTh)) then 将dis上与src对应的圆内的所有PIXEL的辉度值设定为src的上述圆周上的PIXEL的平均辉度值.

else NOP // do nothing

注意, 若dis中的PIXEL已被设过值(非0),则进行两项平均.-----这种平均法在数学上是很不严密的,但对相近的数值之间误差不会很大.

**方法2:**



设r为圆的半径,以r为间隔在图像上定出作为圆心的PIXEL.

以上述PIXEL为圆心, 分别求半径为r的**圆周上[注1]**的图像值**直方图h1**和**圆内**的图像值**直方图h2**.

~~找出两个整数序列~~**~~h1, h2~~**~~之间的长度为~~*~~l~~* **~~[注2]~~**~~的子段间最大匹配.~~

~~比如有以下两整数数列A和B~~

~~A 9801120044067990000~~

~~B 00711201990006~~

~~例如,~~

~~若求~~*~~l~~*~~=4的最佳匹配, 则匹配的平均差为0-----因为A,B皆存在一个连续的长度为4的数字段“1120” ----- 100%匹配!~~

~~若求~~*~~l~~*~~=6的最佳匹配, 则匹配的平均差肯定不为0-----~~

~~有两对最佳匹配段:~~

~~A的“112004” 与B的 “112019” ----- 匹配的平均差 = 6/~~*~~l~~* ~~= 1~~

~~A的“990000” 与B的 “990006” ----- 匹配的平均差 = 6/~~*~~l~~* ~~= 1~~

但实际的匹配测度要使用VECTOR之间的相似度(Pearson's product-moment coefficient)--------见<<七． **二値化方法**>>中的说明.

即, 对**h1**和**h2,**在**直方图**的各个不同的位置(并列,同时地)进行长度为*l*的匹配计算(相似度计算).

从左右两端同时开设取长度为*l*的匹配计算,如果进行到所剩长度不足*l*处则停止计算.----可并行地进行!

最后,检查由以上计算得到的相似度中的最大值是否大于外部指定的一个阈值rh.

这时的**匹配差**的定义为 **1－相似度**

**~~上述匹配问题,可用字符串(string)及子字符串(sub string)的并行匹配技术来实现-----~~**~~用不同的THREAD进行不同位置上的匹配即可.~~

-------因为现在变成了等长的整数列,直接使用VECTOR之间的相似度(Pearson's product-moment coefficient)即可.

if **匹配差** > matchErrTh // float

将dis的对应该圆内的PIXEL值全置为圆周上的图像值的中值平均**[注3],** 若dis中的PIXEL已被设过值(非0),则进行两项平均----

这样进行平均在数学上是很不合理的,但当预计诸值比较接近时作为近似计算是可以的.

**注1:** 与中心PIXEL的距离跟指定半径相差为4以内的点皆作为该圆周上的点.

“半径为相差4以内” 指的是, 距中心点的距离d,与指定的半径r之间满足下记关系.

**0 < d-r < 4**  🡪 实际计算方法:

int r2 = r\*r; // 一次性求好后不再计算.

int r3 = (r+4)\*(r+4); // 一次性求好后不再计算.

这两个数当由 CALL KERNEL时已经确定, 故可作为KERNEL FUNCTION的参数代入.

--------------------------

即, 定义

\_\_global\_\_ ker (…, int r2, int r3,….) {

…

int d = (x-x0)(x-x0) + (y-y0)(y-y0); // (x0, y0): 指定的圆心坐标.

if (d > r2 && d < r3) // 🡨 原始的条件转化为

…

}

Call

ker(…, r2, r3, ….);

-------------------------

这样便避免了开根计算及浮点小数的计算,比较.

**~~注2:~~** *~~l~~*~~是间接给出的,即是由参数float~~ *~~percent~~* ~~∈~~~~(0, 1]来算出的:~~ *~~l~~* ~~=~~ *~~percent~~* **~~\*~~**~~圆周上的点数~~

**注3:** 排序后分别去掉两端25%的部分, 对剩下的50%的圆周上的PIXEL的图像值求平均.

圆周半径int r, 阈值 float matchErrTh , ~~以及长度参数float~~ *~~percent~~*皆为函数的参数

Q1：为什么处理后的图像要保存在shared memory中呢？shared memory大小具有一定的限制，无法保存较大的图像。如果要保存在shared memory中，是否要求只申请一个线程块呢？

A1. 这个问题太并行化了---- 各个处理之间无关,可以任意分割,但处理是对TEST图像的指定部分(由坐标集来指定)进行的,处理结果也是直接反应到TEST图像上的.

中间结果应在shared memory上的吧? -----具体应采用什么MEMORY策略请大家定!

Q2：这里计算与中心PIXEL的距离是否也是按照公式进行计算，其中a，b表示中心PIXEL的坐标，x，y表示其他点坐标？

A2. YES.

Q1: 在需求中提到了两种算法：方法1和方法2

if (此方差<指定的值) 这个指定的值是外部给的一个阈值，方法2中也提到if **匹配差** > rh 这个rh也是外部给的一个阈值，请问这两个外部给的阈值是不是指的同一个阈值？

A: 当然不是一回事. 请见原文的**匹配差**的定义为 **1－相似度**

详见这次寄去的<<理工大軟件学院----第一段階開発項目[ver2.0](2012-10-27).docx >>

Q2: 方法2中：将dis的对应该圆内的PIXEL值全置为圆周上的图像值的中值平均**[注3]，**中值平均是取中间50%的灰度值做平均，

方法1中：将dis上与src对应的圆内的所有PIXEL的辉度值设定为src的上述圆周上的PIXEL的平均辉度值.这个没有标注，就是不需要取中间50%的灰度值做平均，我的理解是这个意思吗？为何第一个不需要但是第二个需要这样做？

A: 多谢思考! 两者不一样----

**方法一**决定了圆周上的图像值分布不会有太大的分散,故简单平均一下便可以了; 而

**方法二**是考察圆内外的灰度特征的区别, 圆周上的图像值可能是较分散的,故取其中值平均.

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

十三.  **ROBUST EDGE DETECTION (RED)**

**----------由两个对向域间的差分确定边缘---------注意, 广义地说, 差分可以是任何统计量之间的,未必只限于图像的灰度值.**

3×3时的例子



unsigned char DI[ ][ ]

Q1：请问河边老师，对于需求分析中，边缘检测算法中“对指定的一个坐标SET,按其X,Y坐标值的范围开辟相应的GRID SIZE(DIMENSION); 并用2D 硬寻址各个相应的PIXEL值”可不可以不用硬寻址，而是在处理每个像素点的时候计算这些领域内的坐标索引，因为texture硬寻址现在效果不是很好，如果采用每次计算时计算这些领域内的坐标，可以不用开辟三维结构。

A1 : 当然可以,速度至上! 只是对2D硬寻址有些遗憾的感觉, 之前我对此还是抱有较大期望的……

“texture硬寻址现在效果不是很好”是必然的呢, 还是特定型号的GPGPU所特有的现象?---------河边

Q2：请问河边老师，需求中说为了抵抗较大的噪音纹理,对现场测试图像来说,对向领域的SIZE = 11\*11比较适当.那么在实际编码中，我们是否固定模板的SIZE为11×11，还是模板可选。

A2：由给出的公式可以推倒出各种SIZE的模板, 但为了效率起见, 我们可以预先制作好3\*3, 5\*5, 7\*7, 9\*9, 11\*11

这5个模板的坐标SET,放到GPU MEMORY中,其它可不考虑.

Q3： int\* guidingset参数，原文注释是EDGE抽出的位置坐标----即只对该坐标SET所规定的位置上进行EDGE-----，这个guidingset参数在我们理解有两种含义：一是检测出的边缘存入一个点集，即guidingset，二是guidingset是一个指导区域，我们只对这个点集在图像中标记的区域内进行边缘检测的操作。我们想问到底哪种理解正确？

A3： guidingset 是怎么来的? 原文没有说明背景,抱歉!

在实际应用中,图像内容往往复杂,质量又不好,故为了得到安定有效的EDGE,常需要人的指导-----即由人画出一个较粗糙的EDGE线,在该线附近进行自动的EDGE检出处理.

So the answer is, 以距guidingset中的最近点距离为指定值以内的带状域(guidingset)为对象,搜索真实的EDGE位置.

------这个带状域是如何生成的? 请参照"**九**"中的那个**绿色**的带状域.

Q4：请问河边老师，需求中说的对辉度值小于小值端5%处的值的PIXEL值设为小值端5%处的值(min)，5%指的是小值的5%还是小值端位置的5%？

A4：假如有100个数据,将它们排好序, 那么

从最小值端开始COUNT, 假如第5个数据为a, 则 小值端5%处的值(min)= a. 同样,

从最大值端开始COUNT,若第5个数据为B, 则 大值端5%处的值(max)= B.

Q5：请问河边老师，两个个数最多的图像值之差MMD与minInterval这个参数的具体作用是？两个个数最多的图像值作何解释？

A5: MMD: 图像直方图中的第一大峰值和第二大峰值所对应的图像值之差. 由于直方图形状的不规则,不理想性,两个峰值之间在距离上应有一定的间隔限制,这就是minInterval这个参数的作用.

如何判定局部PEAK值(峰值)----有信用性,是最大的难点….请大家考虑一下---不要太复杂的方法. 注意,很多图像的HIST并无明显的PEAK !

mixtureGaussian法很典型,但计算量太大-----不知可否通过并列计算克服其弱点?

mixtureGaussian法是一个很有意义的挑战! ----若成功,很多地方都可用到!

第3步, 即所谓的**Non**-**maximum Suppression,** 其思想与CANNY算法是一样的.

这种检查是并行的----使差分结果IMAGE(DI)中的每一个位置(PIXEL)都对应一个THREAD,

各个THREAD在其对应的DP号所指示的方向上检查DI , 考察是否有比自己所具有的差分值更大的值.

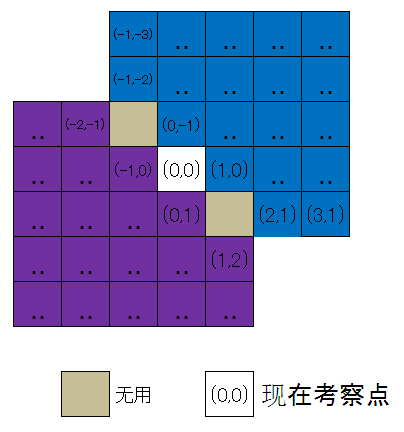
如果有,则使该THREAD所对应的DI上的某位置处的值为0.

即, GPU THREAD GRID

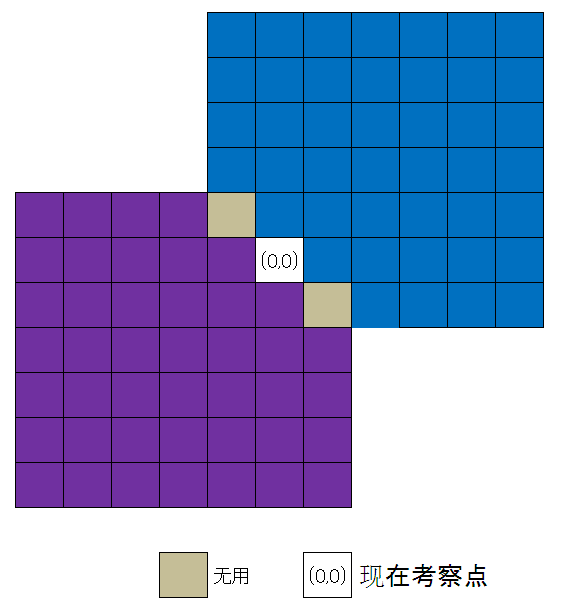
这里给出的是3×3差分单位的情况,类似地,

发展到n×n的差分计算单位的情况

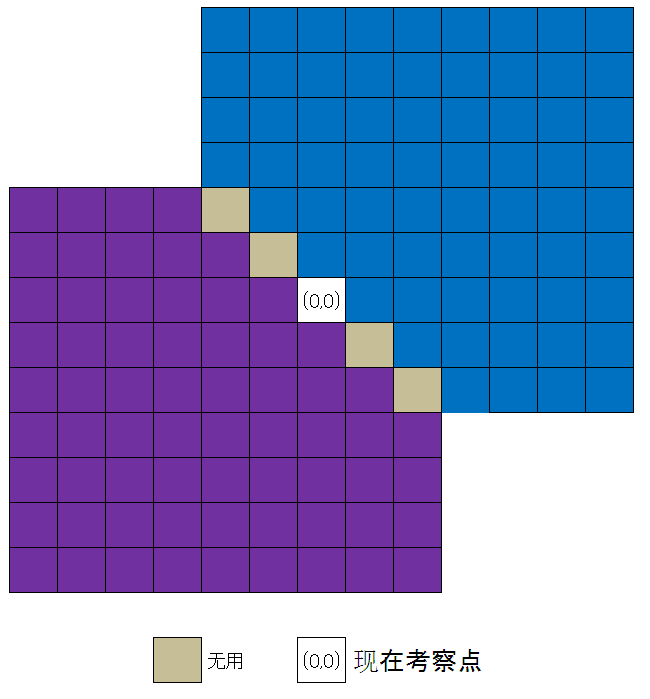
一般地, 需要分别确定两个**对向领域**的坐标SET-----可镜像地由一方的SET得到另一方的SET.



5×5



7×7



9×9

设两个对向领域的大小各为m×m, 它们之间的**重叠部分**的大小为n×n

-----这里只允许 m,n皆为奇数.

则m,n间存在如下关系:

n = (m+1)/2 or

n = (m-1)/2

取上述算式中保证n为奇数的那个算式.

所以, 5×5领域间的重叠部分的大小为3×3, 7×7领域间的重叠部分的大小也为3×3;

而9×9领域间的重叠部分的大小为5×5, 11×11领域间的重叠部分的大小也为5×5;

为了抵抗较大的噪音纹理,对现场测试图像来说, **对向领域**的SIZE = 11×11比较适当.

同样, 还应确定其它3种对向关系的坐标集对.

**这里的ROBUST EDGE DETECTION (RED)有2种算法**

1. **单纯平均法**-----如上面的例3×3、5×5、….直接对向差分.

int detectEdgeSA ( Image\* image, // 作为处理对象的图像

int diffsize, // 3: 3×3; 5: 5×5; 7: 7×7; …..

CoordiSet\* guidingset,

float\* percent,

float complex ); // EDGE抽出的位置坐标----即只对该坐标SET所规定的位置上进行EDGE检出处理-----

若guidingset==NULL, 则对整个image的ROI进行EDGE检出处理.　　 complex值一般为0.1 – 3.0)

detectEdgeSA(Image\* image, CoordiSet\* duidingset) 是RobustEdgeDetection CLASS 的MEMBER METHOD.

其内若还要用到其它参数,都是RobustEdgeDetection CLASS 的property,可由相应的

set…, get… method 设定或取得-----详见最新的<<CODING RULES>>

1. **特征VECTOR法**

int detectEdgeFV ( Image\* image, // 作为处理对象的图像

int diffsize, // 3: 3×3; 5: 5×5; 7: 7×7; …..

CoordiSet\* guidingset,

float\* percent,

float complex ,

int window ); // EDGE抽出的位置坐标----即只对该坐标SET所规定的位置上进行EDGE检出处理-----

若guidingset==NULL, 则对整个image的ROI进行EDGE检出处理. 　　complex值一般为0.1 – 3.0)

detectEdgeFV(Image\* image, CoordiSet\* duidingset) 是RobustEdgeDetection CLASS 的MEMBER METHOD,

其内若还要用到其它参数,都是RobustEdgeDetection CLASS 的property,可由相应的

set…, get… method 设定或取得-----详见最新的<<CODING RULES>>

**对指定领域求下记3种量**(MAV, MMD, SGM) **----**特征VECTOR

1. 中值平均MAV----将领域里的辉度值**排序**;
2. 对辉度值小于小值端5%处的值的PIXEL值设为小值端5%处的值(min);
3. 对辉度值大于大值端5%处的值的PIXEL值设为大值端5%处的值(max);
4. 对小值端5%处以上和大值端5%处以下的中部辉度求平均(占全DATA量的90%); 🡪 此乃 “中值平均MAV”
5. **两个**个数最多的图像值之差MMD----要先对各个**辉度的出现次数**进行统计; 但这两个峰值之间的最小距离是由参数int minInterval来控制的.

如何判定局部PEAK值----有信用性,是最大的难点….请大家考虑一下---不要太复杂的方法. 注意,很多图像的HIST并无明显的PEAK!

作为一个简单的方法,对指定宽度的窗口内的HIST累积即可-----在HIST的各个位置上进行这个窗口内的累计计算.

找出最大的一个, 并在距具有最大累积量的窗口(int window)中心为指定距离(minInterval)以上的地方(左右)开始,寻找第二大窗口累积量.

上述第一,第二大累积量,即作为第一,第二大PEAK值.

1. 求两个对向域的图像值的标准差SGM---σ

A. 求指定坐标集所规定的**整个图像**的MAV, MMD, SGM-----相当于各量的期望值, 分别记为EMAV, EMMD, ESGM(应预先求好---只求一次即可).

B.

(1) 对PIXEL p ---相当于**对向**图中的(0,0)点, 分别求其两个对向域的MAV, MMD, SGM 3个量:

MAV1, MMD1, SGM1;

若guidingset != NULL, 则上述诸处理所涉及的对象,只限于guidingset所规定的坐标SET.

在处理上,以guidingset为出发点,通过遍历guidingset中的坐标来进行上述各种处理即可.

具体地, 开辟一个与guidingset中的各个坐标相对应的GPU THREAD, 分别同时对各个THREAD对应的坐标处的PIXEL进行上述特征VECTOR以及4个方向的差分计算……

MAV2, MMD2, SGM2

(2) 进行如下的计算,得到正规化的量值:

NMAV1p = MAV1 / EMAV

NMMD1p =MMD1 / EMMD

NSGM1p = SGM1 / ESGM

NMAV2p = MAV2 / EMAV

NMMD2p =MMD2 / EMMD

NSGM2p = SGM2 / ESGM

上述诸量皆为正.

C. 求两个对向域的上述3个特征量间的几何距离

disp = 255\*root{ [(NMAV1p – NMAV2p )2 + mmdWeight\*(NMMD1p – NMMD2p )2 + sgmWeight \*(NSGM1p – NSGM2p )2 ] / wsm } 四舍五入.

wsm = 1 + mmdWeight + sgmWeight;

float mmdWeight, sgmWeight∈[0, 1000] 为两个参数-----范围这么大的目的是为了可以任意保留某些差分成分,而无视另一些差分成分.

将disp记入到前述的unsigned char DI buffer中与p相应的位置.

~~if disp > 255 then disp=255;~~

D.　用与3×3时的例子相同的方法,搜索DI中各个方向上的最大值点.

**注意,** int minInterval, float mmdWeight, float sgmWeight 皆为RobustEdgeDetection CLASS 的property,

可由相应的set…, get… method 设定或取得-----详见最新的<<CODING RULES>>

int detectEdgeSA(…), detectEdgeFV(…)

经上记差分处理及Non-maximum Suppression处理后,生成一个原始的EDGE MATRIX, 对原始的EDGE MATRIX进行下记精化处理,得到最终的EDGE 坐标.

float per = \*percent; //(在无信息时, \*percent的初始值一般设为0.5)

int n = 2\*width\*height\*complex; // width, height : 图像中EDGE检查范围的宽和长.

(1) 计算**EDGE点强度**的直方图

(2) 从**EDGE点强度**最大值端开始, 计算EDGE点强度直方图的**累积直方图**.

设**累积直方图**上的最大HISTOGRAM值为maxh;

(3) 在**累积直方图**上从**EDGE点强度**最大值端开始搜索(以下,搜索或移动的STEP皆为1),

(4) 在**累积直方图**上找到HISTOGRAM值最接近maxh\* per 处的EDGE强度m;

(5) 对最初的EDGE点检出结果,保留EDGE强度>m 的EDGE点-----生成一个关于EDGE点的2值图像;

(6) 对上述2值图像用5\*5 SIZE的morphological structuring element 进行DILATION处理(并行),

然后进行THINNING处理(并行)------这两个并行处理必须用以前的开发成果.

(7) 统计THINNING处理后的EDGE点数c;

(8) if (c/n < 0.9) {

per + = 0.1;

在**累积直方图**上由m向EDGE点强度**小**的方向移动, goto (4);

}

else if (c/n > 1.1) {

per – = 0.1;

在**累积直方图**上由m向EDGE点强度**大**的方向移动, goto (4);

}

else {

\*percent = per;

return 0;

}

求应选取的大强度的EDGE数占EDGE总数的比例.

Q3：说明中仅给出了斜向重叠的示例，但却没有给出纵向（DP=0）或横向（DP=1）重叠示例，不知这两种方向如何完成重叠？

A3：已给出了size = 3×3时的所有4个方向的差分PATTERN的分布样子, 其他SIZE只有DPn = 3 时的例子. 但DPn = 0,1,2 时对位方向与size = 3×3时是完全一样的. 而重叠部分大小的计算公式已经给出.

Q4：“3x3时的例子”下方的说明方框（里面有3条），存在如下疑问：

1. 该算法的输入信息是图像（密矩阵表示）还是坐标数组（抑或稀疏矩阵或模板）?

A4-1) : 问得好! 很抱歉, 原文的这部分表达得乱! 已经重写,请见添付文件.

1. 没有理解河边老师所说的“2D硬寻址”是何意？

A4-2): “2D硬寻址”是GPGPU的一种功能,记得是对TEXTURE MEMORY有效,详细请请教王宇新老师或小于博士.

1. “DPn”与“DP”的区别不清楚？

A4-3): DPn 即DP的方向号----请见原文的注释的方框中的说明.

1. 第3步要求在DP方向上进行“一定范围的搜索”，“一定范围”是通过参数给定？

A4-4) : 这个“一定范围”也是RobustEdgeDetection CLASS 的property之一, 由相应的

set…, get… method 设定或取得-----与N-neighbor是同一概念----一个int型的正整数:

int searchScope;

1. 第3步最后所述“消除自己”没有理解，是指用最大值代替当前值，还是直接将当前值归零？

A4-5) : 此目的即所谓的**Non**-**maximum Suppression,** 其思想与CANNY算法是一样的.

这种检查是并行的----使差分结果IMAGE(DI)中的每一个位置(PIXEL)都对应一个THREAD, 各个THREAD在其对应的DP号所指示的方向上检查DI , 考察是否有比自己所具有的差分值更大的值.

如果有,则使该THREAD所对应的DI上的某位置处的值为0.

Q5：“9x9”下面的蓝色字部分，说m和n之间的关系，但知何时应该取n=(m+1)/2，又何时当取n=(m-1)/2？

A5 : 原文中明确指出 “取上述算式中保证n为奇数的那个算式” **!**

Q6：“特征VECTOR方法”存在如下问题：

1. 第1步的“领域里”指“3x3”“5x5”这样的当前点的邻域中？

A6-1 首先对整个指定的领域求MAV, MMD, SGM三种量,作为这三种量的期望值----分别记为,

EMAV, EMMD, ESGM(应预先求好---只求一次即可).

然后对两个对向域分别求这3个量

1. 第4步“两个最大的图像值之差”中“图像值”指“灰度值”？如果存在三个灰度并列第一时，应该如何处理？

A6-2): 精确的说法应是 “两个个数最多的图像值之差”;

在图像处理领域,“图像值”一般都是指“灰度值”----color的情况为3维的量.

“如果存在三个灰度并列第一应该如何处理？”: **此问题提得好! 谢谢!!**

如果存在若干种图像值,它们的个数之差不足图像总的PIXEL数的1/N,皆为HISTOGRAM的局部PEAK值, 则取两个图像值相差最大的PEAK之间的图像值之差. (N在此为外界指定参数)

如何判定局部PEAK值----有信用性,是最大的难点….请大家考虑一下---不要太复杂的方法. 注意,很多图像的HIST并无明显的PEAK!

1. 第5步，不理解“分散”为何意？

A6-3): 严格的叫法应为”标准偏差([standard](http://ejje.weblio.jp/content/standard) deviation)” -----统计学上常用SIGMA表示,基本算式如下:

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

**十四．简易的显著域DETECTION**

【思想】

若要检查大小为N-size的異物,应先用**优势画素値法[**参见”**二.**1/N缩小图像的生成”]

将原画像1/N缩小,

之后再将缩小图像N倍扩大---从而得到背景画像.

用原画像减背景画像;

对此差分图像进行isseki-clustering[参见 ”十一. Local clustering (scale level 別)” ],以及bilateral filtering处理[NVIDIA LIB可用]----

之后对主流画像值以外的诸画像值进行

1. 连通域检出
2. 连通域的合并

----最后确定符合指定面积，长短径要求的连通域。

求出坐标SET给定的OBJECT的HISTOGRAM上的前N个PEAK的位置,

用第一PEAK以外的PEAK位置所对应的图像辉度值进行图像分割(即找连通域----用前次成果),

对面积在指定范围内的连通域求最小外接有向矩形(中心坐标,转角,长轴的长度,短轴的长度----最小外接有向矩形可用上次的开发成果)

【步骤】

1. 将原OBJECT画像进行1/N缩小 (用**优势画素値法[**参见”**二.**1/N缩小图像的生成”])
2. 对1/N缩小图像进行广义的MEDIAN处理[if 有指定此功能的参数存在]

3. 再将缩小的图像进行双线性INTERPOLATION得到N倍扩大---作为背景画像

4. 用原画像减背景画像(差分图像生成)

5. 对4的差分图像进行isseki-clustering[参见 ”十一. Local clustering (scale level 別)” ]

6. 对5的结果进行bilateral filtering处理[if 有指定此功能的参数存在][注]

7. ~~对6的结果图像求HISTOGRAM上的前N个PEAK的位置,~~

~~并用第一PEAK以外的PEAK位置所对应的图像辉度值进行图像分割(即检出连通域----用前次成果)~~

对6的结果图像进行0 - 255 LEVEL的图像灰度分割(即,并行地检出连通域----用前次成果)

8. 对7的结果,进行CLOSING处理(用前次成果)---- in order of 连通域的合并. (CLOSING的构造元素的SIZE由外部指定)

之后,以面积与总面积之比大于阈值者为背景,其它则根据由参数所指定的要求,进行面积计算或形状特征的计算.

9. 对符合面积要求或形状特征的连通域, 求其最小外接有向矩形(中心坐标,转角,长轴的长度,短轴的长度----最小外接有向矩形可用上次的开发成果),

并将这些量设定到RETURN VECTOR中.

~~9. 返回长轴的长度,短轴的长度乃至方向符合指定范围的连通域中心坐标,转角,长轴的长度,短轴的长度.~~

[注]　这里的参数,指的是bilateral filtering处理涉及到的3个参数----空间σs , rangeσr , 和进行filtering处理的次数 repeat.

float ss, sr; // 空间σs , rangeσr

int repeat;

**十五．Hough変換** (for 直線and 円) ( 河边 参考C++ CODE) 【参照 NVIDIA LIB可能】

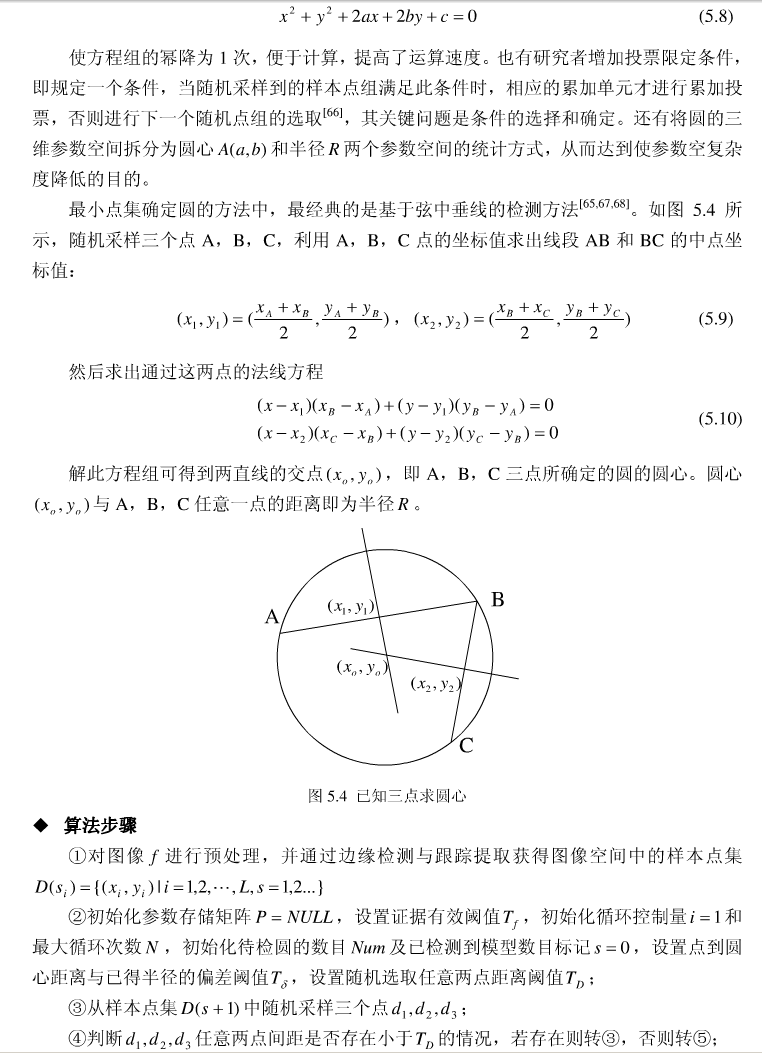
**1. 直线的Hough检出法(最小二乘法可不做)**

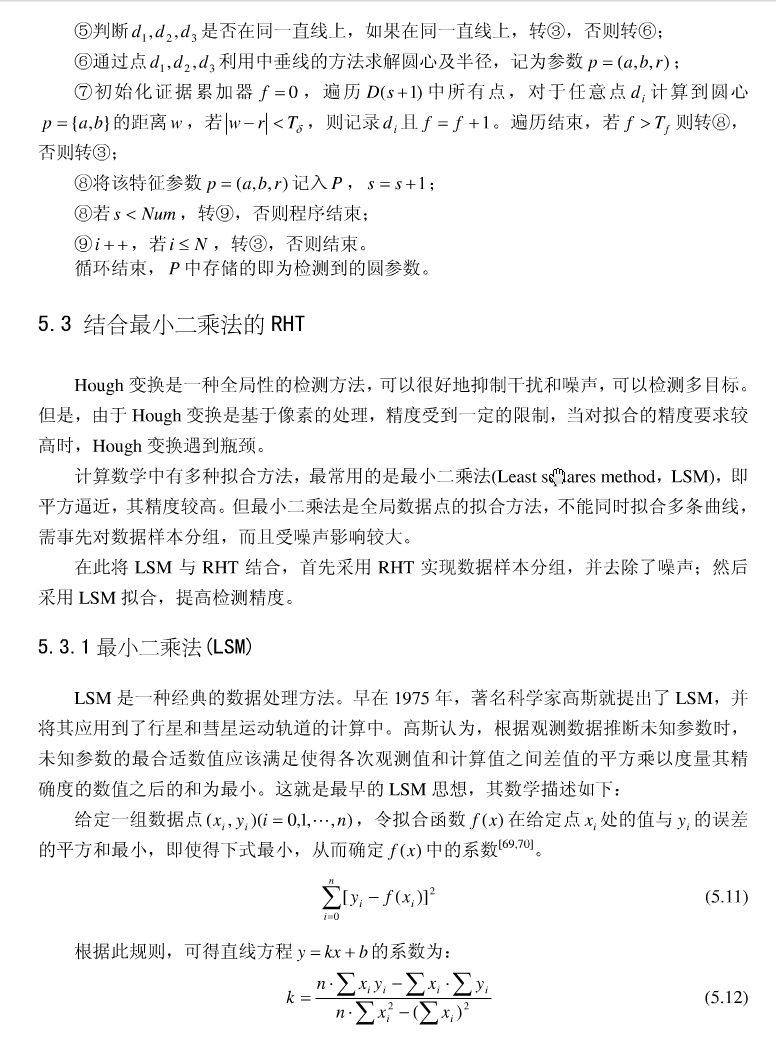
(1) 线的倾角的RESOLUTION(即角度统计的单位的大小)作为参数由外部给出

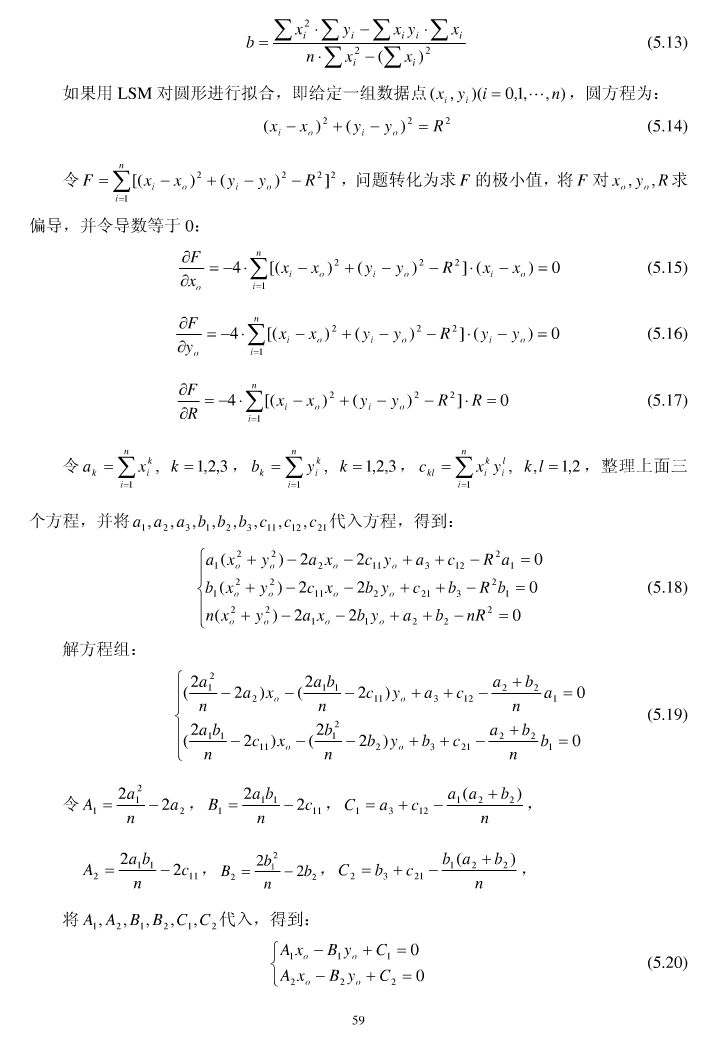
(2) 所要求得的直线的个数作为参数由外部给出

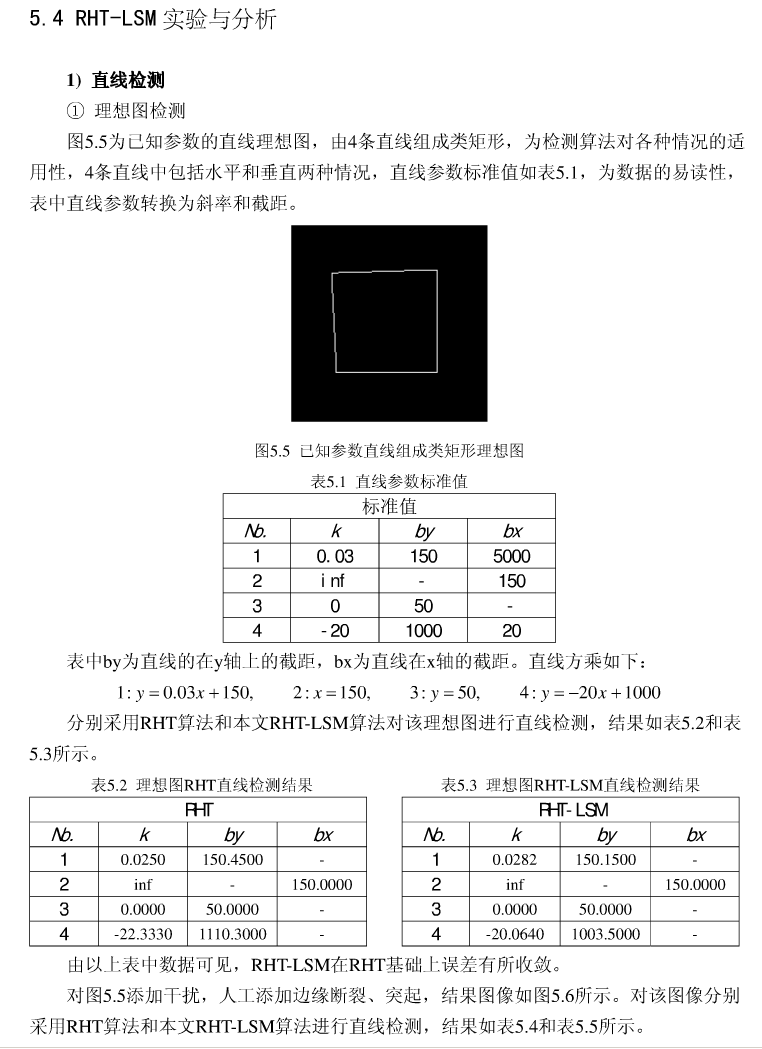
**2. 圆的Hough检出法**----推荐下述经典方法**(最小二乘法可不做)**

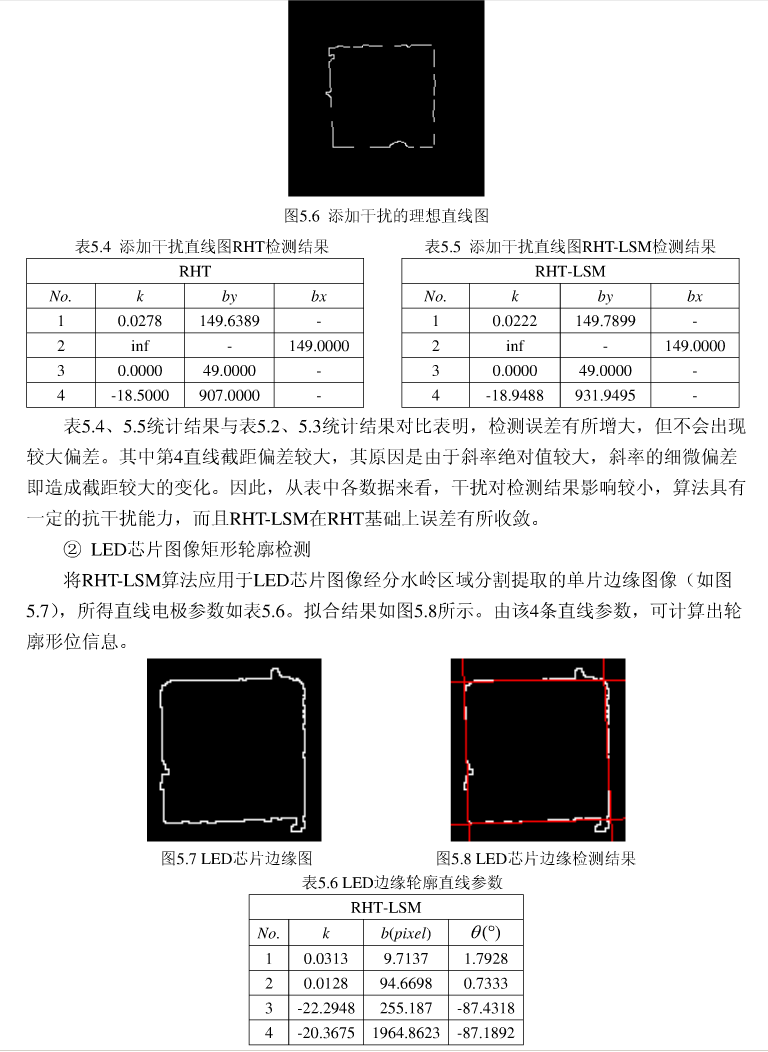
**3. 矩形的Hough检出法**----推荐下述经典方法

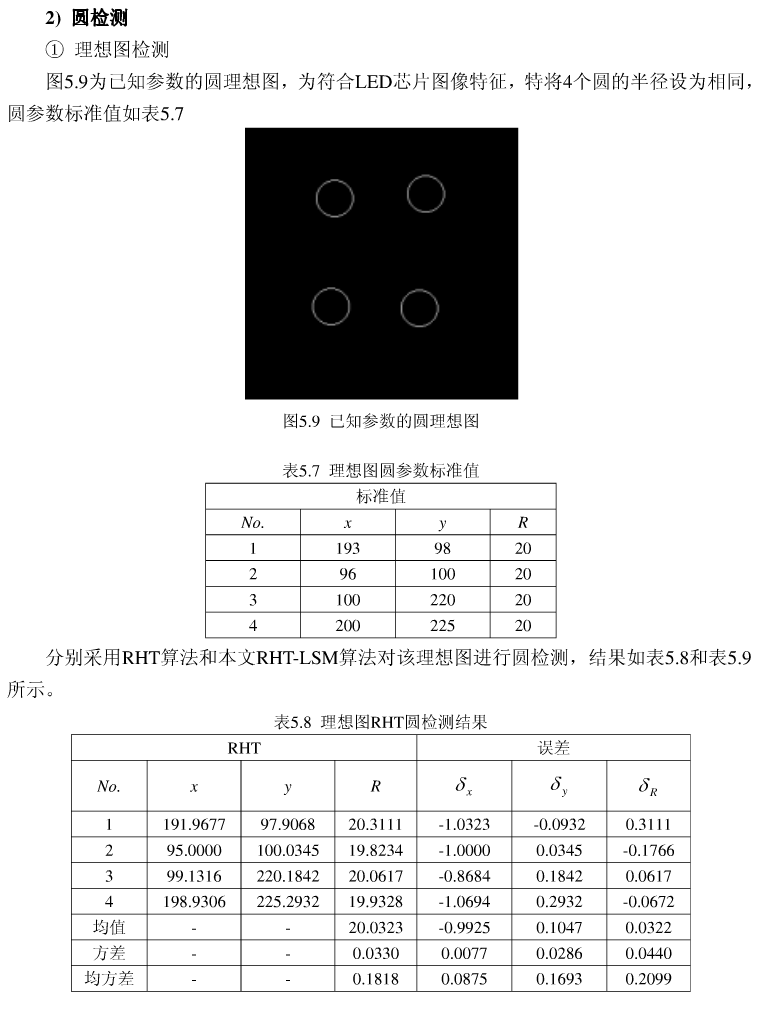
****

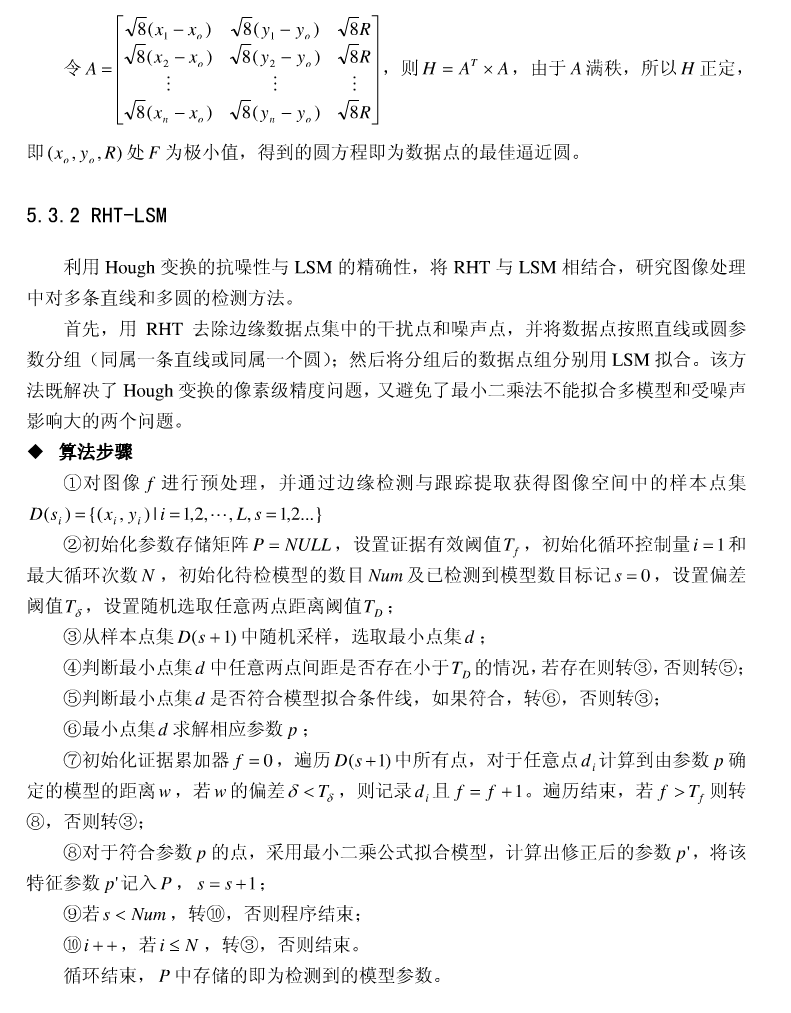
****

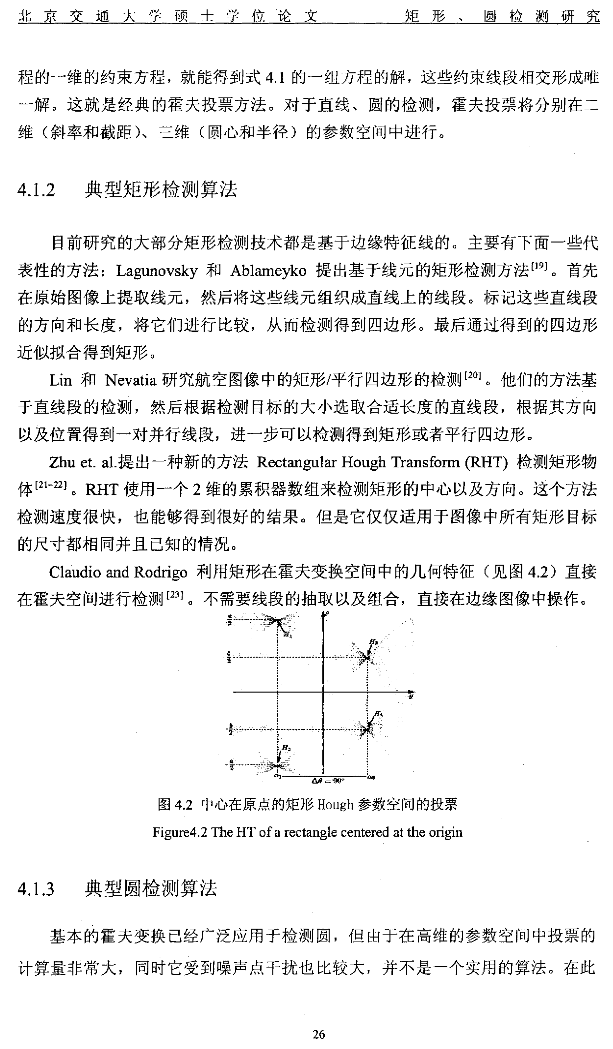
****

****

****

****

****

****

“Rectangle detection based on a windowed Hough transform ”

The problem of detecting rectangular structures in images arises in many applications, from building extraction in aerial images to particle detection in cryo-electron microscopy. This paper proposes a new technique for rectangle detection using a windowed Hough transform. Every pixel of the image is scanned, and a sliding window is used to compute the Hough transform of small regions of the image. Peaks of the Hough image (which correspond to line segments) are then extracted, and a rectangle is detected when four extracted peaks satisfy certain geometric conditions. Experimental results indicate that the proposed technique produced promising results for both synthetic and natural images.

**This paper appears in:**  
[Computer Graphics and Image Processing, 2004. Proceedings. 17th Brazilian Symposium on](http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=9360)  
**Date of Conference:** 17-20 Oct. 2004  
**Author(s):** [Jung, C.R.](http://ieeexplore.ieee.org/search/searchresult.jsp?searchWithin=p_Authors:.QT.Jung,%20C.R..QT.&newsearch=partialPref)   
Ciencias Exatas e Tecnologicas, Univ. do Vale do Rio dos Sinos, Sao Leopoldo, Brazil   
[Schramm, R.](http://ieeexplore.ieee.org/search/searchresult.jsp?searchWithin=p_Authors:.QT.Schramm,%20R..QT.&newsearch=partialPref)   
**Page(s):** 113 - 120   
**Product Type:** Conference Publications

Q1：文档中提到的参考C++ CODE没有看到。

A1: 请稍等一下, 我将给出我以前的直线的“Hough变换”CODE.

Q2：Hough变换的最小二乘法（LSM）是不是可以不用实现？

A2: yes.

Q3：Hough变换的输入、输出参数的有没有什么具体要求。

A3: 问得好.

对直线的检出问题:

1. 要检出的直线的最大个数;
2. 要检出的直线的角度范围; // 角度以从X轴的正方向开始成逆时针关系为正.
3. 最小角度单位(度); // bin size for counter.
4. 图像的POINTER: image\* imgx;
5. 有效坐标集: int\* guidingset // if guidingset ==NULL then 以整个imgXXX 的ROI中的PIXEL为直线的检出对象; else 以guidingset 中的坐标为直线的检出对象.

Q4：可否给出文档中所示的参考文献的具体信息。

A4： 关于直线的“Hough变换”,我所用的是最基本的方法----教科书或网上到处都是 ----我已记不得多少年前的文献了;

而为检出圆的Hough变换,以及为检出四方形的Hough变换, 参考资料我已经加在上次的<< 大連理工大軟件学院----第一段階開発項目[ver1.0---修改后](2012-6-26).docx>> 中了,

但是还应有不少好方法, 如果大家发现了更好的方法, 请提出来!

# Hough变换---Q/A

1. Hough变换的输入图像是不是经过边缘提取的二值图像？

A: 是的. 有效值(unsigned char)由外部指定.

1. 对于开发方的需求：在做Hough变换时，会有一个输入的点集guidingset，若该点集为空，则Hough变换就处理输入图像的ROI区域；若该点集不为空，则只处理该点集。不知道这样的理解和您的需求是否一致。

A: 是的. 若guidingset不为空, 则统计的对象只限定于guidingset中的坐标----直接对guidingset中的坐标进行遍历即可.

1. 对于Hough变换检测直线，上次给出的输入参数中有一个最小角度单位，不知道是什么意思？是否指的是检测直线时，一次的角度增量Δθ。

A: 是的.

1. Hough变换的输出，只用得到输出图像，还是还要得到检测的直线的参数数据。

A: 将检测到的直线或圆以参数的形式返回.　具体返回形式为:

for 直线---- typedef LineParam

flaot angle; // 与横向的夹角---逆时针为正,顺时针为负.

int distance; // 与坐标原点的距离---四舍五入只取整.

long votes; // 得票数

for 圆---- typedef CircleParam

int x,y ; // 圆心坐标.

int radius; // 半径长---四舍五入只取整.

long votes; // 得票数

for 矩型---- typedef RectParam

int xc,yc ; // 几何中心坐标.

int x1,y1, x2,y2, x3,y3, x4,y4; // 4个顶点的坐标-----按顺时针排放.

long votes; // 得票数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:

将上述3种数组定义成3种数据型(数据结构), 以各个数据结构(型)的array的形式返回不同的结果.

-----这3种数据型(数据结构)最好定义成Hough变换CLASS内部的,不必要GLOBAL化.

十七. **精确的MAGNITUDE OF GRADIENT 计算法( PMG)**

1. 用采用**对向域**差分法計算gradientの强度(magnitude)-----共4个对向方向, **对向域**size選択可能-------具体见, **<<ROBUST EDGE DETECTION >>**
2. 选取差分绝对值最大的gradient a,设其左右两个近邻的gradient (原始差分)分别b, c. -----b,c間肯定成45°角
3. 按下记算法计算**magnitude的平方(**assuming a is the largest magnitude of gradient in 4 – directions; b,c are its neighbors)

int\*\* md;

int a, b, c, x, y;

for all pixel (x,y) of indicated region(represented by a coordinate set):

// 以下, a,b,c分别代表gradient a, b, c 的magnitude (**对向域**差分值的绝对值)

if (|b-c|/a < t) md[y][x] = a\*a; // t: threshold,外部参数

else if (b > c) {

int s=a+b;

}

else {

int s=a+c;

}

注：

|b-c|/a < t ⇒ (b ≒ c) or (a >>b and a>> c)

十八. **ROBUST REGION MATCHING ( RRM)**

对**PMM**的结果图像进行的匹配.

匹配算法与<<**经由坐标映射的图像相关计算>>**中的【**具有坐标映射关系的图像间的相关系数计算**】相同

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆ END /◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/

**備忘**for第二阶段开发内容

1. GRADIENT FLOW-------由此统计OBJECT的倾斜-------一个或两个主要角度
2. OBJECT的中线检出-------及对其进行HOUGH变换. 事先应适当地将图像缩小
3. 求指定两个OBJECT间所成角度----可用1,2的结果
4. 两个指定OBJECT之间的连接程度
5. 指定角点处的角度
6. 指定范围内指名OBJECT的个数
7. 对指名的OBJECT进行下记指标的检查
8. 与本TEST图像上的另一个指名object间的最小距離是否在指定范围内
9. 与本TEST图像上的另一个指名object间的中心距離是否在指定范围内
10. 与本TEST图像上的另一个指名object间的中心连线的相对于WORK方向的角度是否在指定范围内
11. 空间分布指标 (长轴,短轴,角度(相对于WORK方向), 面积) 是否在指定范围内
12. 与样本图像上的指名object的形状相关系数;
13. 与样本图像上的指名object的HISTOGRAM的相关系数;
14. 与样本图像上的指名object的HU MOMENT 指标间的误差;
15. 与样本图像上的指名object的空间分布指标 (长轴,短轴,方向,面积) 间的误差

**一. 　交叉区域检查**

1. 对ROI上的作为显著区域的连通域RC*i* , 在对应的TOI上寻找对应的连通域-----摄动匹配.

若正规相关系数 < 指定的值, 在TI上与RC*i*相对应的位置上画一个椭圆---其长短(直)径同RCi的外接最小有向矩形.

1. 对TOI求出作为显著区域的连通域TCi , 若TCi存在, 则在对应的ROI上寻找对应的连通域-----摄动匹配.

若正规相关系数 < 指定的值, 在TI的TC*i*位置上画一个椭圆--- TC*i*的外接最小有向矩形的内接(椭)圆.

**二.　 局部轮廓检查**

**三.　OBJECT群的匹配及位置指标计算**

**四.　边缘欠损(欠け)或偏离检出**

首先用RED进行边缘EDGE检出, 并求出EDGE中值.

对EDGE值 < EDGE中值的p%的EDGE点进行标识.

对EDGE值 < EDGE最大值的q%的EDGE点进行标识.

**五.** 给出两个OBJECT的坐标SET, 当这两个OBJECT在TEST图像上的的中心位置已知时,

求这两个OBJECT之间的最小距离(即最近边缘之间的距离)以及两中心之间的距离.

**六.丝状物等具有特殊形状特征的OBJECT检出**

1. 丝状物―――学習でHu moment vectorを取得
2. 粒状物―――学習でHu moment vectorを取得
3. 棒状物―――学習でHu moment vectorを取得
4. 凸状物―――学習か指標設定
5. 凹状物―――学習か指標設定
6. 不规则分布---指标设定

与指的HU MOMENT 诸标间的误差;

1. 与样本图像上的指名object的形状相关系数;
2. 与样本图像上的指名object的HISTOGRAM的相关系数;
3. 与样本图像上的指名object的HU MOMENT 指标间的误差;
4. 与样本图像上的指名object的空间分布指标 (长轴,短轴,方向,面积) 间的误差
5. 其它形状指标
6. 特定形状的几何体-----圆,多变形

严格----全匹配

柔软----MOMENT----回转,SCALAR敏感型;回转,SCALAR非敏感型;

1. 与样本图像上的指名object的形状相关系数;
2. 与样本图像上的指名object的HISTOGRAM的相关系数;
3. 与样本图像上的指名object的HU MOMENT 指标间的误差;
4. 与样本图像上的指名object的空间分布指标 (长轴,短轴,方向,面积) 间的误差
5. 其它形状指标
6. 任何显著域(突变,ムラ、ソバカス、こんたみ)-----面积等
7. pattern検査
8. 边缘欠损(欠け), 出っ張り(bridge)
9. 边缘変形(輪郭変形, 偏离)

指名object: 分**特指**和**泛指**

## 特指 列举一个或几个object的名字, 名字特征为尾部带数字--- 如, “poly7”, “poly23”, “ellipse34”,…

**泛指** 列举一个或几个object的类型名--- 如, “poly”, “ellipse”

通过匹配的方法确定TEST图像上各个指名object的坐标set

◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/◆/