cvWatershed 函数分析

1. 分水岭算法原理简介
分水岭算法是一种基于拓扑理论的数学形态学的分割方法，其基本思想是把图像看作是测地学上的拓扑地貌，图像中每一点像素的灰度值表示该点的海拔高度，每一个局部极小值及其影响区域称为集水盆，而集水盆的边界则形成分水岭。分水岭的概念和形成可以通过模拟浸入过程来说明。在每一个局部极小值表面，刺穿一个小孔，然后把整个模型慢慢浸入水中，随着浸入的加深，每一个局部极小值的影响域慢慢向外扩展，在两个集水盆汇合处构筑大坝，即形成分水岭，将不同区域分割开来。

一般的分水岭算法会对微弱边缘，图像中的噪声，物体表面细微的灰度变化造成过度的分割。[OpenCV]中的分水岭算法 cvWatershed 对此进行了改进，它使用预定义的一组标记来引导对图像的分割。其入口参数有两个，第一个是待分割的图像，第二个是标记图像。其实现过程即是以标记图像为参照，对待分割图像进行像素处理，构建分水岭，并将结果保存在标记图像中输出。

对该函数的理解，**关键在于理解标记函数的来历及其特点，此外，关注在构建分水岭时的具体代码实现方法。**

前面提到，opencv中的分水岭算法利用标记图像来防止出现过分割的情况，标记图像的作用是为算法指明图像中大致可以分为几个区域，并用不同的数值来表示不同的区域，算法则在已知的区域中开始灌水，知道分水岭构建成功。

2. 例程
```
#include <iostream>

#include <opencv2\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>

using namespace std;
using namespace cv;


int main(int argc, char **argv)
{
    string fileName = "D:/Document/VS workplace/barCode/srcImage/timg.jpg";
    Mat src = imread(fileName);
    imshow("src", src);
```

`![](index_files/322513721.png)`

```
Mat gray;
cvtColor(src, gray, CV_BGR2GRAY);

Mat threod;
//二值化图像，初步提取前景图
threshold(gray, threod, 60, 255, CV_THRESH_BINARY);
imshow("threshold", threod);
![](index_files/322650362.png)
```

```
//腐蚀二值化后的图像，得到前景图
Mat fg;
erode(threod, fg, Mat(), Point(-1, -1), 8);
imshow("fg", fg);
![](index_files/322770327.png)
```

//对二值化图像进行膨胀，得到背景图，注意，这里背景用 灰度值128表示，二值化方法也使用了反转

```
Mat bg;
dilate(threod, bg, Mat(), Point(-1, -1), 8);
threshold(bg, bg, 1, 128, THRESH_BINARY_INV);
imshow("bg", bg);
![](index_files/322992894.png)
```

利用前景图像和背景图像进行 + 运算，得到标记图像。从标记图像中可以看出，图中有两个区域，128（背景）和255（前景），而0值则是不确定的区域，相当于区域和区域之间的边界，后续的算法中将在这个不确定区域中构建出单像素宽度的分水岭。

```
Mat marker = fg + bg;
imshow("marker", marker);
marker.convertTo(marker, CV_32S);
![](index_files/323054561.png)
```

```
//调用 Opencv中的算法 cvWatershed，结果保存在标标记图像中。
cvWatershed(src, marker);
Mat result;
//需要进行数据类型转换才能显示
marker.convertTo(result, CV_8U);
imshow("result", result);
![](index_files/323292431.png)
```

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

3.cvWatershed 函数实现

```cpp
//节点数据结构定义，一个节点表示一个像素
typedef struct CvWSNode
{
    struct CvWSNode* next; //下一个节点
    int mask_ofs; //该像素在标记图像中的位置
    int img_ofs; //该像素在原图像中的位置
}
CvWSNode;
//队列定义，用来表示属于同一区域的像素点的集合
typedef struct CvWSQueue
{
    CvWSNode* first; //指向队列第一个节点
    CvWSNode* last; //指向队列最后一个节点
}
CvWSQueue;
//分配内存空间，用来存放节点数据
static CvWSNode*
icvAllocWSNodes( CvMemStorage* storage )
{
    CvWSNode* n = 0;

    int i, count = (storage->block_size - sizeof(CvMemBlock))/sizeof(*n) - 1;

    n = (CvWSNode*)cvMemStorageAlloc( storage, count*sizeof(*n) );
    for( i = 0; i < count-1; i++ )
        n[i].next = n + i + 1;
    n[count-1].next = 0;

    return n;
}


CV_IMPL void
cvWatershed( const CvArr* srcarr, CvArr* dstarr )
```

```
{
   const int IN_QUEUE = -2; //用来标记像素已经进入队列
   const int WSHED = -1; //标记分水岭像素
   const int NQ = 256; //定义队列的数量，最多256个（灰度值范围）
   cv::Ptr<CvMemStorage> storage;

   CvMat sstub, *src;
   CvMat dstub, *dst;
   CvSize size;
   CvWSNode* free_node = 0, *node;
   CvWSQueue q[NQ];
   int active_queue;
   int i, j;
   int db, dg, dr;
   int* mask;
   uchar* img;
   int mstep, istep;
   int subs_tab[513];

   // MAX(a,b) = b + MAX(a-b,0)
   #define ws_max(a,b) ((b) + subs_tab[(a)-(b)+NQ])
   // MIN(a,b) = a - MAX(a-b,0)
   #define ws_min(a,b) ((a) - subs_tab[(a)-(b)+NQ])
   //压入队列操作，idx 为队列号，mofs为像素在标记图像中的位置，iofs为像素在原图像中的位置
   #define ws_push(idx,mofs,iofs)  \
   {                              \
      if( !free_node )            \
         free_node = icvAllocWSNodes( storage );\
      node = free_node;           \
      free_node = free_node->next;\
      node->next = 0;             \
      node->mask_ofs = mofs;      \
      node->img_ofs = iofs;       \
      if( q[idx].last )           \
         q[idx].last->next=node; \
      else                        \
         q[idx].first = node;    \
      q[idx].last = node;         \
   }
   //出队列操作，参数同上
```

```
#define ws_pop(idx,mofs,iofs)   \
{                               \
    node = q[idx].first;        \
    q[idx].first = node->next;  \
    if( !node->next )           \
        q[idx].last = 0;        \
    node->next = free_node;     \
    free_node = node;           \
    mofs = node->mask_ofs;      \
    iofs = node->img_ofs;       \
}
//计算两个像素值得差，结果保存在 diff中，ptr 为像素指针
//这里比较的是三通道 RGB 像素，计算每个通道的差值，返回差值最大的。
#define c_diff(ptr1,ptr2,diff)      \
{                                   \
    db = abs((ptr1)[0] - (ptr2)[0]);\
    dg = abs((ptr1)[1] - (ptr2)[1]);\
    dr = abs((ptr1)[2] - (ptr2)[2]);\
    diff = ws_max(db,dg);           \
    diff = ws_max(diff,dr);         \
    assert( 0 <= diff && diff <= 255 ); \
}

src = cvGetMat( srcarr, &sstub );
dst = cvGetMat( dstarr, &dstub );

if( CV_MAT_TYPE(src->type) != CV_8UC3 )
    CV_Error( CV_StsUnsupportedFormat, "Only 8-bit, 3-channel input images are supported" );

if( CV_MAT_TYPE(dst->type) != CV_32SC1 )
    CV_Error( CV_StsUnsupportedFormat,
        "Only 32-bit, 1-channel output images are supported" );

if( !CV_ARE_SIZES_EQ( src, dst ))
    CV_Error( CV_StsUnmatchedSizes, "The input and output images must have the same size" );

size = cvGetMatSize(src);
storage = cvCreateMemStorage();

istep = src->step;
```

```
img = src->data.ptr;
mstep = dst->step / sizeof(mask[0]);
mask = dst->data.i;

memset( q, 0, NQ*sizeof(q[0]) );

for( i = 0; i < 256; i++ )
    subs_tab[i] = 0;
for( i = 256; i <= 512; i++ )
    subs_tab[i] = i - 256;

// draw a pixel-wide border of dummy "watershed" (i.e. boundary) pixels
//边界处理为分水岭
for( j = 0; j < size.width; j++ )
    mask[j] = mask[j + mstep*(size.height-1)] = WSHED;

// initial phase: put all the neighbor pixels of each marker to the ordered queue -
// determine the initial boundaries of the basins
//循环将所有与标记区域紧挨的位置区域像素压入队列
for( i = 1; i < size.height-1; i++ )
{
    img += istep; mask += mstep;
    mask[0] = mask[size.width-1] = WSHED;

    for( j = 1; j < size.width-1; j++ )
    {
        int* m = mask + j;
        if( m[0] < 0 ) m[0] = 0;
        //当前像素是未知区域像素，且其位置紧挨标记区域
        if( m[0] == 0 && (m[-1] > 0 || m[1] > 0 || m[-mstep] > 0 || m[mstep] > 0) )
        {
            uchar* ptr = img + j*3;//原图形是三通道，所以*3
            //以下计算出队列号 idx
            int idx = 256, t;
            if( m[-1] > 0 )
                c_diff( ptr, ptr - 3, idx );
            if( m[1] > 0 )
            {
                c_diff( ptr, ptr + 3, t );
                idx = ws_min( idx, t );
```

```
            }
            if( m[-mstep] > 0 )
            {
                c_diff( ptr, ptr - istep, t );
                idx = ws_min( idx, t );
            }
            if( m[mstep] > 0 )
            {
                c_diff( ptr, ptr + istep, t );
                idx = ws_min( idx, t );
            }
            assert( 0 <= idx && idx <= 255 );
            ws_push( idx, i*mstep + j, i*istep + j*3 );//将该像素在 标记图像和原图像中的位置信息压
如队列
            m[0] = IN_QUEUE;
        }
    }
}

// find the first non-empty queue
for( i = 0; i < NQ; i++ )
    if( q[i].first )
        break;

// if there is no markers, exit immediately
if( i == NQ )
    return;

active_queue = i;
img = src->data.ptr;
mask = dst->data.i;
//循环处理队列里的数据
// recursively fill the basins
for(;;)
{
    int mofs, iofs;
    int lab = 0, t;
    int* m;
    uchar* ptr;
```

```
if( q[active_queue].first == 0 )
{
    for( i = active_queue+1; i < NQ; i++ )
        if( q[i].first )
            break;
    if( i == NQ )
        break;
    active_queue = i;
}
//出队列
ws_pop( active_queue, mofs, iofs );
//分别定位到指定位置像素
m = mask + mofs;
ptr = img + iofs;
t = m[-1];
if( t > 0 ) lab = t;
t = m[1];
if( t > 0 )
{
    if( lab == 0 ) lab = t;
    else if( t != lab ) lab = WSHED;
}
t = m[-mstep];
if( t > 0 )
{
    if( lab == 0 ) lab = t;
    else if( t != lab ) lab = WSHED;
}
t = m[mstep];
if( t > 0 )
{
    if( lab == 0 ) lab = t;
    else if( t != lab ) lab = WSHED;
}
assert( lab != 0 );
//得出该像素值，如果左右或上下都是已标记区域，则该像素就是分水岭
m[0] = lab;
if( lab == WSHED )
    continue;
//当前像素处理了后，要判断与之紧挨的位置是否有未知区域像素，有则压入队列
```

```
        if( m[-1] == 0 )
        {
            c_diff( ptr, ptr - 3, t );
            ws_push( t, mofs - 1, iofs - 3 );
            active_queue = ws_min( active_queue, t );
            m[-1] = IN_QUEUE;
        }
        if( m[1] == 0 )
        {
            c_diff( ptr, ptr + 3, t );
            ws_push( t, mofs + 1, iofs + 3 );
            active_queue = ws_min( active_queue, t );
            m[1] = IN_QUEUE;
        }
        if( m[-mstep] == 0 )
        {
            c_diff( ptr, ptr - istep, t );
            ws_push( t, mofs - mstep, iofs - istep );
            active_queue = ws_min( active_queue, t );
            m[-mstep] = IN_QUEUE;
        }
        if( m[mstep] == 0 )
        {
            c_diff( ptr, ptr + istep, t );
            ws_push( t, mofs + mstep, iofs + istep );
            active_queue = ws_min( active_queue, t );
            m[mstep] = IN_QUEUE;
        }
    }
}
```
灰度图像版本
下面的函数是从 opencv 分水岭算法修改而来，用于处理输入图像是灰度图像的情况，其中主要是修改 像素之间灰度差值的宏定义。
```cpp
#include <iostream>

#include <opencv2\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>
```

```
using namespace std;
using namespace cv;


typedef struct CvWSNode
{
    struct CvWSNode* next;
    int mask_ofs;
    int img_ofs;
}
CvWSNode;

typedef struct CvWSQueue
{
    CvWSNode* first;
    CvWSNode* last;
}
CvWSQueue;

CvWSNode* icvAllocWSNodes(CvMemStorage* storage)
{
    CvWSNode* n = 0;

    int i, count = (storage->block_size - sizeof(CvMemBlock)) / sizeof(*n) - 1;

    n = (CvWSNode*)cvMemStorageAlloc(storage, count*sizeof(*n));
    for (i = 0; i < count - 1; i++)
        n[i].next = n + i + 1;
    n[count - 1].next = 0;

    return n;
}
void Watershed(InputArray __src, InputOutputArray __markers)
{

**    CvMat __srcarr = __src.getMat(), __dstarr = __markers.getMat();
    CvMat *srcarr = &__srcarr, *dstarr = &__dstarr;**
```

```
    const int IN_QUEUE = -2;
    const int WSHED = -1;
    const int NQ = 256;
    cv::Ptr<CvMemStorage> storage;

    CvMat sstub, *src;
    CvMat dstub, *dst;
    CvSize size;
    CvWSNode* free_node = 0, *node;
    CvWSQueue q[NQ];
    int active_queue;
    int i, j;
**   //int db, dg, dr;**
    int* mask;
    uchar* img;
    int mstep, istep;
    int subs_tab[513];

    // MAX(a,b) = b + MAX(a-b,0)
#define ws_max(a,b) ((b) + subs_tab[(a)-(b)+NQ])
    // MIN(a,b) = a - MAX(a-b,0)
#define ws_min(a,b) ((a) - subs_tab[(a)-(b)+NQ])

#define ws_push(idx,mofs,iofs)  \
    {                            \
    if (!free_node)            \
    free_node = icvAllocWSNodes(storage); \
    node = free_node;           \
    free_node = free_node->next; \
    node->next = 0;             \
    node->mask_ofs = mofs;      \
    node->img_ofs = iofs;       \
    if (q[idx].last)           \
    q[idx].last->next = node; \
        else                    \
        q[idx].first = node;    \
        q[idx].last = node;        \
    }

#define ws_pop(idx,mofs,iofs)   \
```

```
    {                                  \
    node = q[idx].first;         \
    q[idx].first = node->next;  \
    if (!node->next)            \
    q[idx].last = 0;         \
    node->next = free_node;     \
    free_node = node;           \
    mofs = node->mask_ofs;      \
    iofs = node->img_ofs;       \
    }

**#define c_diff(ptr1,ptr2,diff)      \
    {                                  \
    diff = abs((ptr1)[0] - (ptr2)[0]); \
    assert(0 <= diff && diff <= 255); \
    }**

    src = cvGetMat(srcarr, &sstub);
    dst = cvGetMat(dstarr, &dstub);

**    if (CV_MAT_TYPE(src->type) != CV_8UC1)
        CV_Error(CV_StsUnsupportedFormat, "Only 8-bit, 1-channel input images are supported");**

    if (CV_MAT_TYPE(dst->type) != CV_32SC1)
        CV_Error(CV_StsUnsupportedFormat,
        "Only 32-bit, 1-channel output images are supported");

    if (!CV_ARE_SIZES_EQ(src, dst))
        CV_Error(CV_StsUnmatchedSizes, "The input and output images must have the same size");

**    size.height = src->rows;
    size.width = src->cols;**

    storage = cvCreateMemStorage();

    istep = src->step;
    img = src->data.ptr;
    mstep = dst->step / sizeof(mask[0]);
    mask = dst->data.i;
```

```
memset(q, 0, NQ*sizeof(q[0]));

for (i = 0; i < 256; i++)
    subs_tab[i] = 0;
for (i = 256; i <= 512; i++)
    subs_tab[i] = i - 256;

// draw a pixel-wide border of dummy "watershed" (i.e. boundary) pixels
for (j = 0; j < size.width; j++)
    mask[j] = mask[j + mstep*(size.height - 1)] = WSHED;

// initial phase: put all the neighbor pixels of each marker to the ordered queue -
// determine the initial boundaries of the basins
for (i = 1; i < size.height - 1; i++)
{
    img += istep; mask += mstep;
    mask[0] = mask[size.width - 1] = WSHED;

    for (j = 1; j < size.width - 1; j++)
    {
        int* m = mask + j;
        if (m[0] < 0) m[0] = 0;
        //当前像素是未知像素，且其上下左右至少有一个是标记像素
        if (m[0] == 0 && (m[-1] > 0 || m[1] > 0 || m[-mstep] > 0 || m[mstep] > 0))
        {
            uchar* ptr = img + j;//原图形是三通道，所以*3
            int idx = 256, t;
            if (m[-1] > 0)
                c_diff(ptr, **ptr - 1**, idx);
            if (m[1] > 0)
            {
                c_diff(ptr, **ptr + 1**, t);
                idx = ws_min(idx, t);
            }
            if (m[-mstep] > 0)
            {
                c_diff(ptr, ptr - istep, t);
                idx = ws_min(idx, t);
            }
            if (m[mstep] > 0)
```

```
            {
                c_diff(ptr, ptr + istep, t);
                idx = ws_min(idx, t);
            }
            assert(0 <= idx && idx <= 255);
            ws_push(idx, i*mstep + j, i*istep + j);//将该像素在 标记图像和原图像中的位置信息压如队
列
            m[0] = IN_QUEUE;
        }
    }
}

// find the first non-empty queue
for (i = 0; i < NQ; i++)
if (q[i].first)
    break;

// if there is no markers, exit immediately
if (i == NQ)
    return;

active_queue = i;
img = src->data.ptr;
mask = dst->data.i;

// recursively fill the basins
for (;;)
{
    int mofs, iofs;
    int lab = 0, t;
    int* m;
    uchar* ptr;

    if (q[active_queue].first == 0)
    {
        for (i = active_queue + 1; i < NQ; i++)
        if (q[i].first)
            break;
        if (i == NQ)
            break;
```

```
        active_queue = i;
    }

    ws_pop(active_queue, mofs, iofs);

    m = mask + mofs;
    ptr = img + iofs;
    t = m[-1];
    if (t > 0) lab = t;
    t = m[1];
    if (t > 0)
    {
        if (lab == 0) lab = t;
        else if (t != lab) lab = WSHED;
    }
    t = m[-mstep];
    if (t > 0)
    {
        if (lab == 0) lab = t;
        else if (t != lab) lab = WSHED;
    }
    t = m[mstep];
    if (t > 0)
    {
        if (lab == 0) lab = t;
        else if (t != lab) lab = WSHED;
    }
    assert(lab != 0);
    m[0] = lab;
    if (lab == WSHED)
        continue;

    if (m[-1] == 0)
    {
**          c_diff(ptr, ptr - 1, t);
        ws_push(t, mofs - 1, iofs - 1);**
        active_queue = ws_min(active_queue, t);
        m[-1] = IN_QUEUE;
    }
    if (m[1] == 0)
```

```
        {
**            c_diff(ptr, ptr + 1, t);
            ws_push(t, mofs + 1, iofs + 1);**
            active_queue = ws_min(active_queue, t);
            m[1] = IN_QUEUE;
        }
        if (m[-mstep] == 0)
        {
            c_diff(ptr, ptr - istep, t);
            ws_push(t, mofs - mstep, iofs - istep);
            active_queue = ws_min(active_queue, t);
            m[-mstep] = IN_QUEUE;
        }
        if (m[mstep] == 0)
        {
            c_diff(ptr, ptr + istep, t);
            ws_push(t, mofs + mstep, iofs + istep);
            active_queue = ws_min(active_queue, t);
            m[mstep] = IN_QUEUE;
        }
    }
}



int main(int argc, char **argv)
{
    string fileName = "D:/Document/VS workplace/barCode/srcImage/timg.jpg";
    Mat src = imread(fileName, CV_LOAD_IMAGE_GRAYSCALE);
    imshow("src", src);

    Mat gray = src.clone();
    //cvtColor(src, gray, CV_BGR2GRAY);

    Mat threod;
    threshold(gray, threod, 60, 255, CV_THRESH_BINARY);
    imshow("threshold", threod);

    Mat fg;
    erode(threod, fg, Mat(), Point(-1, -1), 8);
```

```
    imshow("fg", fg);

    Mat bg;
    dilate(threod, bg, Mat(), Point(-1, -1), 8);
    threshold(bg, bg, 1, 128, THRESH_BINARY_INV);
    imshow("bg", bg);

    Mat marker = fg + bg;
    imshow("marker", marker);
    marker.convertTo(marker, CV_32S);

    Watershed(src, marker);
    Mat result;
    marker.convertTo(result, CV_8U);
    imshow("result", result);




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