

图 6.48 原始图

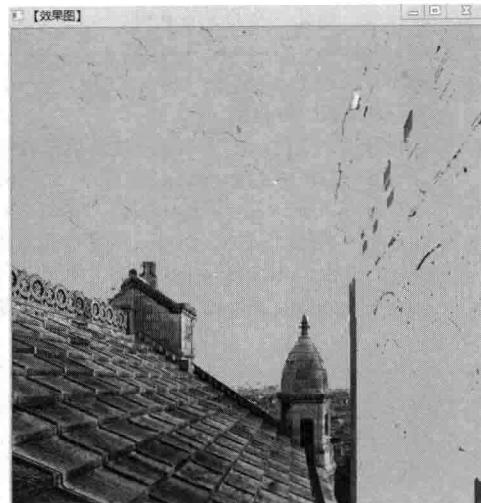


图 6.49 漫水填充效果图

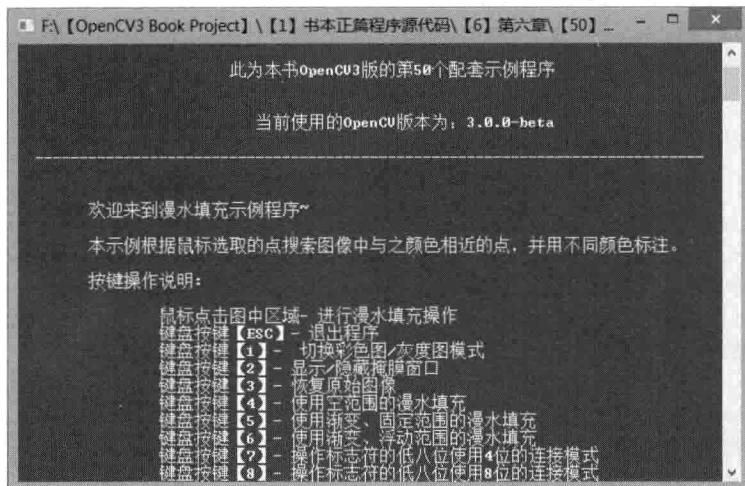


图 6.50 程序操作说明

可以看到，此程序有不少按键功能。我们用鼠标对窗口中的图形进行多次单击，就可以得到类似 PhotoShop 中魔棒的效果。当然，就这短短的两百来行代码写出来的东西，体验还是比不上 PS 的魔棒工具的。程序详细注释的源码如下。

```

//-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----【全局变量声明部分】-----
```

```

#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
using namespace cv;
using namespace std;
```

```

//      描述：全局变量声明
//-----
Mat g_srcImage, g_dstImage, g_grayImage, g_maskImage;//定义原始图、目标
图、灰度图、掩模图
int g_nFillMode = 1;//漫水填充的模式
int g_nLowDifference = 20, g_nUpDifference = 20;//负差最大值、正差最大值
int g_nConnectivity = 4;//表示 floodFill 函数标识符低八位的连通值
int g_bIsColor = true;//是否为彩色图的标识符布尔值
bool g_bUseMask = false;//是否显示掩膜窗口的布尔值
int g_nNewMaskVal = 255;//新的重新绘制的像素值

//-----【onMouse() 函数】-----
//      描述：鼠标消息 onMouse 回调函数
//-----
static void onMouse( int event, int x, int y, int, void* )
{
    // 若鼠标左键没有按下，便返回
    //此句代码的 OpenCV2 版为：
    //if( event != CV_EVENT_LBUTTONDOWN )
    //此句代码的 OpenCV3 版为：
    if( event != EVENT_LBUTTONDOWN )
        return;

    //-----【<1>调用 floodFill 函数之前的参数准备部分】-----
    Point seed = Point(x,y);
    int LowDifference = g_nFillMode == 0 ? 0 : g_nLowDifference;//空范
围的漫水填充，此值设为 0，否则设为全局的 g_nLowDifference
    int UpDifference = g_nFillMode == 0 ? 0 : g_nUpDifference;//空范
围的漫水填充，此值设为 0，否则设为全局的 g_nUpDifference

    //标识符的 0~7 位为 g_nConnectivity, 8~15 位为 g_nNewMaskVal 左移 8 位的值,
    16~23 位为 CV_FLOODFILL_FIXED_RANGE 或者 0。
    //此句代码的 OpenCV2 版为：
    //int flags = g_nConnectivity + (g_nNewMaskVal << 8) +(g_nFillMode
    == 1 ? CV_FLOODFILL_FIXED_RANGE : 0);
    //此句代码的 OpenCV3 版为：
    int flags = g_nConnectivity + (g_nNewMaskVal << 8) +(g_nFillMode ==
    1 ? FLOODFILL_FIXED_RANGE : 0);

    //随机生成 bgr 值
    int b = (unsigned)theRNG() & 255;//随机返回一个 0~255 之间的值
    int g = (unsigned)theRNG() & 255;//随机返回一个 0~255 之间的值
    int r = (unsigned)theRNG() & 255;//随机返回一个 0~255 之间的值
    Rect ccomp;//定义重绘区域的最小边界矩形区域

    Scalar newVal = g_bIsColor ? Scalar(b, g, r) : Scalar(r*0.299 + g*0.587
    + b*0.114);//在重绘区域像素的新值，若是彩色图模式，取 Scalar(b, g, r)；若是灰
度图模式，取 Scalar(r*0.299 + g*0.587 + b*0.114)

    Mat dst = g_bIsColor ? g_dstImage : g_grayImage;//目标图的赋值
    int area;

```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//-----【<2>正式调用 floodFill 函数】-----  
if( g_bUseMask )  
{  
    //此句代码的 OpenCV2 版为：  
    //threshold(g_maskImage, g_maskImage, 1, 128,  
    CV_THRESH_BINARY);  
    //此句代码的 OpenCV3 版为：  
    threshold(g_maskImage, g_maskImage, 1, 128, THRESH_BINARY);  
  
    area = floodFill(dst, g_maskImage, seed, newVal, &ccomp,  
    Scalar(LowDifference, LowDifference, LowDifference),  
    Scalar(UpDifference, UpDifference, UpDifference), flags);  
    imshow( "mask", g_maskImage );  
}  
else  
{  
    area = floodFill(dst, seed, newVal, &ccomp, Scalar(LowDifference,  
    LowDifference, LowDifference),  
    Scalar(UpDifference, UpDifference, UpDifference), flags);  
}  
  
imshow("效果图", dst);  
cout << area << " 个像素被重绘\n";  
}  
  
//-----【main() 函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始  
//-----  
int main( int argc, char** argv )  
{  
    //载入原图  
    g_srcImage = imread("1.jpg", 1);  
  
    if( !g_srcImage.data ) { printf("读取图片 image0 错误~! \n"); return  
false; }  
  
    g_srcImage.copyTo(g_dstImage); //复制源图到目标图  
    cvtColor(g_srcImage, g_grayImage, COLOR_BGR2GRAY); //转换三通道的  
    image0 到灰度图  
    g_maskImage.create(g_srcImage.rows+2, g_srcImage.cols+2,  
    CV_8UC1); //利用 image0 的尺寸来初始化掩膜 mask  
  
    //此句代码的 OpenCV2 版为：  
    //namedWindow( "效果图", CV_WINDOW_AUTOSIZE );  
    //此句代码的 OpenCV2 版为：  
    namedWindow( "效果图", WINDOW_AUTOSIZE );  
  
    //创建 Trackbar  
    createTrackbar( "负差最大值", "效果图", &g_nLowDifference, 255, 0 );  
    createTrackbar( "正差最大值", "效果图", &g_nUpDifference, 255, 0 );
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//鼠标回调函数
setMouseCallback( "效果图", onMouse, 0 );

//循环轮询按键
while(1)
{
    //先显示效果图
    imshow("效果图", g_bIsColor ? g_dstImage : g_grayImage);

    //获取键盘按键
    int c = waitKey(0);
    //判断 ESC 是否按下，若按下便退出
    if( (c & 255) == 27 )
    {
        cout << "程序退出.....\n";
        break;
    }

    //根据按键的不同，进行各种操作
    switch( (char)c )
    {
        //如果键盘"1"被按下，效果图在灰度图，彩色图之间互换
        case '1':
            if( g_bIsColor )//若原来为彩色，转为灰度图，并且将掩膜 mask 所有元素设置为 0
            {
                cout << "键盘" "1" "被按下，切换彩色/灰度模式，当前操作为将【彩色模式】切换为【灰度模式】\n";
                cvtColor(g_srcImage, g_grayImage, COLOR_BGR2GRAY);
                g_maskImage = Scalar::all(0); //将 mask 所有元素设置为 0
                g_bIsColor = false; //将标识符置为 false，表示当前图像不为彩色，而是灰度
            }
            else//若原来为灰度图，便将原来的彩图 image0 再次复制给 image，并且将掩膜 mask 所有元素设置为 0
            {
                cout << "键盘" "1" "被按下，切换彩色/灰度模式，当前操作为将【彩色模式】切换为【灰度模式】\n";
                g_srcImage.copyTo(g_dstImage);
                g_maskImage = Scalar::all(0);
                g_bIsColor = true;//将标识符置为 true，表示当前图像模式为彩色
            }
            break;
        //如果键盘按键"2"被按下，显示/隐藏掩膜窗口
        case '2':
            if( g_bUseMask )
            {
                destroyWindow( "mask" );
                g_bUseMask = false;
            }
            else
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
{  
    namedWindow( "mask", 0 );  
    g_maskImage = Scalar::all(0);  
    imshow("mask", g_maskImage);  
    g_bUseMask = true;  
}  
break;  
//如果键盘按键“3”被按下，恢复原始图像  
case '3':  
    cout << "按键“3”被按下，恢复原始图像\n";  
    g_srcImage.copyTo(g_dstImage);  
    cvtColor(g_dstImage, g_grayImage, COLOR_BGR2GRAY);  
    g_maskImage = Scalar::all(0);  
    break;  
//如果键盘按键“4”被按下，使用空范围的漫水填充  
case '4':  
    cout << "按键“4”被按下，使用空范围的漫水填充\n";  
    g_nFillMode = 0;  
    break;  
//如果键盘按键“5”被按下，使用渐变、固定范围的漫水填充  
case '5':  
    cout << "按键“5”被按下，使用渐变、固定范围的漫水填充\n";  
    g_nFillMode = 1;  
    break;  
//如果键盘按键“6”被按下，使用渐变、浮动范围的漫水填充  
case '6':  
    cout << "按键“6”被按下，使用渐变、浮动范围的漫水填充\n";  
    g_nFillMode = 2;  
    break;  
//如果键盘按键“7”被按下，操作标志符的低八位使用 4 位的连接模式  
case '7':  
    cout << "按键“7”被按下，操作标志符的低八位使用 4 位的连接模式\n";  
    g_nConnectivity = 4;  
    break;  
//如果键盘按键“8”被按下，操作标志符的低八位使用 8 位的连接模式  
case '8':  
    cout << "按键“8”被按下，操作标志符的低八位使用 8 位的连接模式\n";  
    g_nConnectivity = 8;  
    break;  
}  
}  
  
return 0;  
}
```

我们在窗口图片上单击鼠标，就可以给其中不同的区域随机着色。程序功能非常丰富，有鼠标操作和键盘 8 个按键的操作，还可以调滑动条。图 6.52、6.53 所示是上述这两百来行代码勾勒出来的程序的一些运行截图，原图如图 6.51 所示。更多的运行效果请大家自行运行书本配套示例程序进行学习和观赏。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>



图 6.51 原始图



图 6.52 运行截图 (1)



图 6.53 运行截图 (2)

随着鼠标的单击，程序会记下我们的以下操作，如图 6.54 所示。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

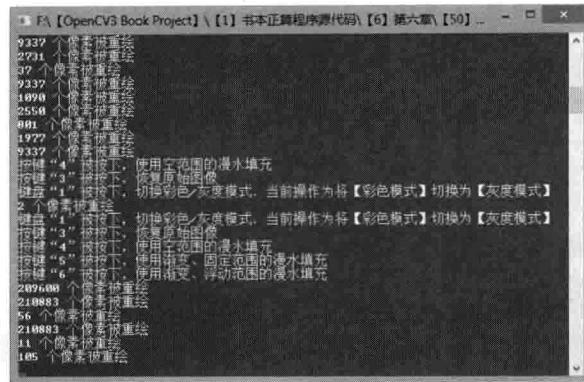


图 6.54 操作记录说明

## 6.6 图像金字塔与图片尺寸缩放

本节中，我们将一起探讨图像金字塔的一些基本概念，学习如何使用 OpenCV 函数 `pyrUp` 和 `pyrDown` 对图像进行向上和向下采样，以及了解专门用于缩放图像尺寸的 `resize` 函数的用法。

### 6.6.1 引言

我们经常会将某种尺寸的图像转换为其他尺寸的图像，如果要放大或者缩小图片的尺寸，笼统来说，可以使用 OpenCV 提供的如下两种方法。

- `resize` 函数。这是最直接的方式
- `pyrUp()`、`pyrDown()` 函数。即图像金字塔相关的两个函数，对图像进行向上采样和向下采样的操作。

`pyrUp`、`pyrDown` 其实和专门用作放大缩小图像尺寸的 `resize` 在功能上差不多，披着图像金字塔的皮，说白了还是在对图像进行放大和缩小操作。另外需要指出的是，`pyrUp`、`pyrDown` 在 OpenCV 的 `imgproc` 模块中的 `Image Filtering` 子模块里，而 `resize` 在 `imgproc` 模块的 `Geometric Image Transformations` 子模块里。

本节，我们将先介绍图像金字塔的原理，接着介绍 `resize` 函数，然后是 `pyrUp` 和 `pyrDown` 函数，最后是一个综合示例程序。

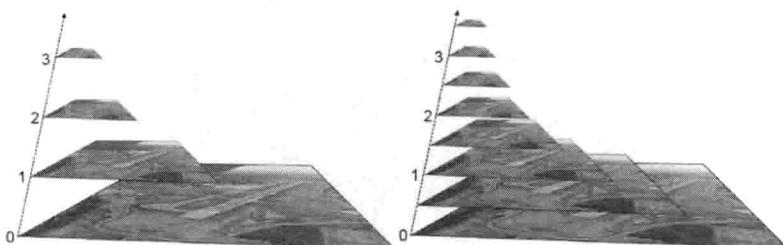
### 6.6.2 关于图像金字塔

图像金字塔是图像中多尺度表达的一种，最主要用于图像的分割，是一种以多分辨率来解释图像的有效但概念简单的结构。

图像金字塔最初用于机器视觉和图像压缩，一幅图像的金字塔是一系列以金字塔形状排列的，分辨率逐步降低且来源于同一张原始图的图像集合。其通过梯次向下采样获得，直到达到某个终止条件才停止采样。

金字塔的底部是待处理图像的高分辨率表示，而顶部是低分辨率的近似。

我们将一层一层的图像比喻成金字塔，层级越高，则图像越小，分辨率越低。如图 6.55、6.56 所示。



6.55 图像金字塔演示图（1）



6.56 图像金字塔演示图（2）

一般情况下有两种类型的图像金字塔常常出现在文献和以及实际运用中。它们分别是：

- 高斯金字塔（Gaussianpyramid）——用来向下采样，主要的图像金字塔。
- 拉普拉斯金字塔（Laplacianpyramid）——用来从金字塔低层图像重建上层未采样图像，在数字图像处理中也即是预测残差，可以对图像进行最大程度的还原，配合高斯金字塔一起使用。

两者的简要区别在于：高斯金字塔用来向下降采样图像，而拉普拉斯金字塔则用来从金字塔底层图像中向上采样，重建一个图像。

要从金字塔第  $i$  层生成第  $i+1$  层（我们将第  $i+1$  层表示为  $G_{i+1}$ ），我们先要用高斯核对  $G_i$  进行卷积，然后删除所有偶数行和偶数列，新得到图像面积会变为源图像的四分之一。按上述过程对输入图像  $G_0$  执行操作就可产生出整个金字塔。

当图像向金字塔的上层移动时，尺寸和分辨率会降低。OpenCV 中，从金字塔中上一级图像生成下一级图像的可以用 `PryDown`，而通过 `PryUp` 将现有的图像在每个维度都放大两遍。

图像金字塔中的向上和向下采样分别通过 OpenCV 的函数 `pyrUp` 和 `pyrDown` 实现。

概括起来就是：

- 对图像向上采样——pyrUp 函数；
- 对图像向下采样——pyrDown 函数。

这里的向下与向上采样，是针对图像的尺寸而言的（和金字塔的方向相反），向上就是图像尺寸加倍，向下就是图像尺寸减半。而如果按图 6.55 和图 6.56 中演示的金字塔方向来理解，金字塔向上图像其实在缩小，这样刚好是反过来了。

但需要注意的是，PryUp 和 PryDown 不是互逆的，即 PryUp 不是降采样的逆操作。这种情况下，图像首先在每个维度上扩大为原来的两倍，新增的行（偶数行）以 0 填充。然后给指定的滤波器进行卷积（实际上是一个在每个维度都扩大为原来两倍的过滤器）去估计“丢失”像素的近似值。

PryDown()是一个会丢失信息的函数。为了恢复原来更高的分辨率的图像，我们要获得由降采样操作丢失的信息，这些数据就和拉普拉斯金字塔有关系了。

### 6.6.3 高斯金字塔

高斯金字塔是通过高斯平滑和亚采样获得一些列下采样图像，也就是说第  $K$  层高斯金字塔通过平滑、亚采样就可以获得  $K+1$  层高斯图像。高斯金字塔包含了一系列低通滤波器，其截止频率从上一层到下一层以因子 2 逐渐增加，所以高斯金字塔可以跨越很大的频率范围。金字塔的图像如图 6.57 所示。

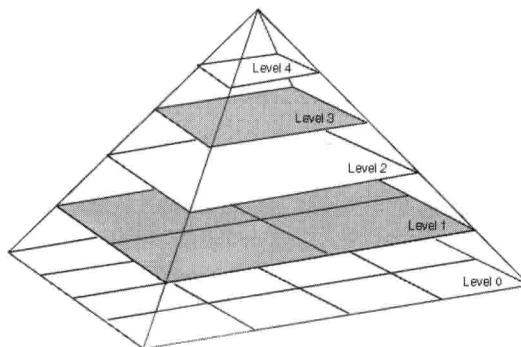


图 6.57 图像金字塔演示图 (3)

另外，每一层都按从下到上的次序编号，层级  $G_{i+1}$  (表示为  $G_{i+1}$  尺寸小于第  $i$  层  $G_i$ )。

#### 1. 对图像的向下取样

为了获取层级为  $G_{i+1}$  的金字塔图像，我们采用如下方法：

- (1) 对图像  $G_i$  进行高斯内核卷积；
- (2) 将所有偶数行和列去除。

得到的图像即为  $G_{i+1}$  的图像。显而易见，结果图像只有原图的四分之一。通过对输入图像  $G_i$  (原始图像) 不停迭代以上步骤就会得到整个金字塔。同时我们也可以看到，向下取样会逐渐丢失图像的信息。

以上就是对图像的向下取样操作，即缩小图像。

## 2. 对图像的向上取样

如果想放大图像，则需要通过向上取样操作得到，具体做法如下。

- (1) 将图像在每个方向扩大为原来的两倍，新增的行和列以 0 填充。
- (2) 使用先前同样的内核（乘以 4）与放大后的图像卷积，获得“新增像素”的近似值。

得到的图像即为放大后的图像，但是与原来的图像相比会发觉比较模糊，因为在缩放的过程中已经丢失了一些信息。如果想在缩小和放大整个过程中减少信息的丢失，这些数据就形成了拉普拉斯金字塔。

接下来一起看一下拉普拉斯金字塔的概念。

### 6.6.4 拉普拉斯金字塔

下式是拉普拉斯金字塔第  $i$  层的数学定义：

$$L_i = G_i - \text{UP}(G_{i+1}) \otimes g_{5 \times 5}$$

式中的  $G_i$  表示第  $i$  层的图像。而 UP() 操作是将源图像中位置为  $(x,y)$  的像素映射到目标图像的  $(2x+1, 2y+1)$  位置，即在进行向上取样。符号  $\otimes$  表示卷积， $g_{5 \times 5}$  为  $5 \times 5$  的高斯内核。

我们下文将要介绍的 pyrUp，就是在进行上面这个式子的运算。

因此，我们可以直接用 OpenCV 进行拉普拉斯运算： $L_i = G_i - \text{PyrUp}(G_{i+1})$

也就是说，拉普拉斯金字塔是通过源图像减去先缩小后再放大的图像的一系列图像构成的。

整个拉普拉斯金字塔运算过程可以通过图 6.58 来概括。

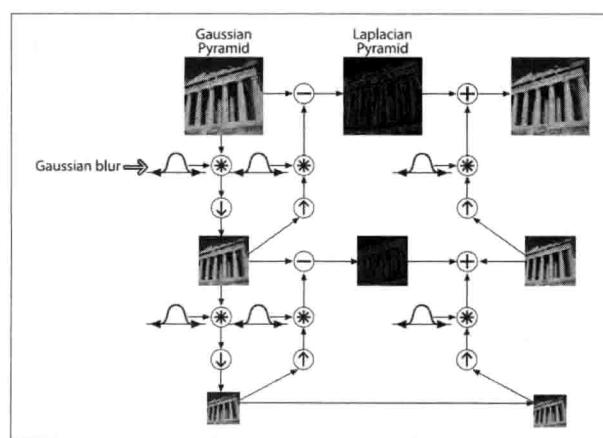


图 6.58 拉普拉斯金字塔运算过程

所以，我们可以将拉普拉斯金字塔理解为高斯金字塔的逆形式。

另外再提一点，关于图像金字塔非常重要的一个应用就是图像分割。图像分割的话，先要建立一个图像金字塔，然后对  $G_i$  和  $G_{i+1}$  的像素直接依照对应的关系，建立起“父与子”关系。而快速初始分割可以先在金字塔高层的低分辨率图像上完成，然后逐层对分割加以优化。

### 6.6.5 尺寸调整：`resize()`函数

`resize()`为 OpenCV 中专门用来调整图像大小的函数。

此函数将源图像精确地转换为指定尺寸的目标图像。如果源图像中设置了 ROI (Region Of Interest，感兴趣区域)，那么 `resize()` 函数会对源图像的 ROI 区域进行调整图像尺寸的操作，来输出到目标图像中。若目标图像中已经设置了 ROI 区域，不难理解 `resize()` 将会对源图像进行尺寸调整并填充到目标图像的 ROI 中。

很多时候，我们并不用考虑第二个参数 `dst` 的初始图像尺寸和类型（即直接定义一个 `Mat` 类型，不用对其初始化），因为其尺寸和类型可以由 `src`、`dsize`、`fx` 和  这几个参数来确定。

看一下它的函数原型：

```
C++: void resize(InputArray src, OutputArray dst, Size dsize, double fx=0,  
double fy=0, int interpolation=INTER_LINEAR )
```

(1) 第一个参数，`InputArray` 类型的 `src`，输入图像，即源图像，填 `Mat` 类的对象即可。

(2) 第二个参数，`OutputArray` 类型的 `dst`，输出图像，当其非零时，有着 `dsize` (第三个参数) 的尺寸，或者由 `src.size()` 计算出来。

(3) 第三个参数，`Size` 类型的 `dsize`，输出图像的大小。如果它等于零，由下式进行计算：

$$\text{dsize} = \text{Size}(\text{round}(fx * \text{src.cols}), \text{round}(fy * \text{src.rows}))$$

其中，`dsize`、`fx`、`fy` 都不能为 0。

(4) 第四个参数，`double` 类型的 `fx`，沿水平轴的缩放系数，有默认值 0，且当其等于 0 时，由下式进行计算：

$$(double)\text{dsize.width/src.cols}$$

(5) 第五个参数，`double` 类型的 `fy`，沿垂直轴的缩放系数，有默认值 0，且当其等于 0 时，由下式进行计算：

$$(double)\text{dsize.height/src.rows}$$

(6) 第六个参数，`int` 类型的 `interpolation`，用于指定插值方式，默认为 `INTER_LINEAR` (线性插值)。

可选的插值方式如下：

- `INTER_NEAREST`——最近邻插值

- INTER\_LINEAR——线性插值（默认值）
- INTER\_AREA——区域插值（利用像素区域关系的重采样插值）
- INTER\_CUBIC——三次样条插值（超过  $4 \times 4$  像素邻域内的双三次插值）
- INTER\_LANCZOS4——Lanczos 插值（超过  $8 \times 8$  像素邻域的 Lanczos 插值）

若要缩小图像，一般情况下最好用 CV\_INTER\_AREA 来插值；而若要放大图像，一般情况下最好用 CV\_INTER\_CUBIC（效率不高，慢，不推荐使用）或 CV\_INTER\_LINEAR（效率较高，速度较快，推荐使用）。

关于插值，我们看几张图就能更好地理解。先看原图（图 6.59）。



图 6.59 原始图

当进行 6 次图像缩小接着 6 次图像放大操作后，两种不同的插值方式得到的效果图如图 6.60、图 6.61 所示。

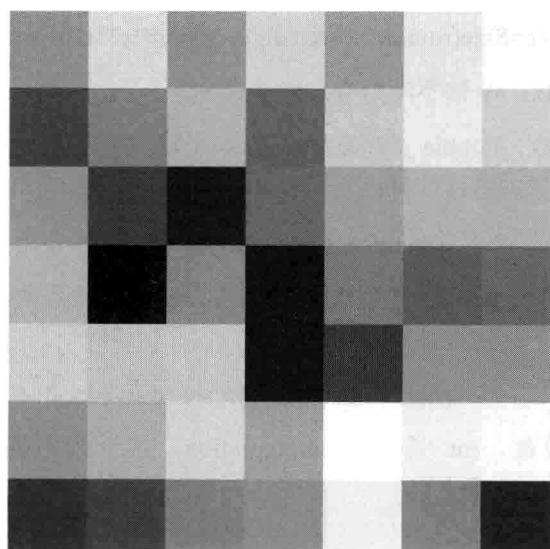


图 6.60 插值效果图（1）

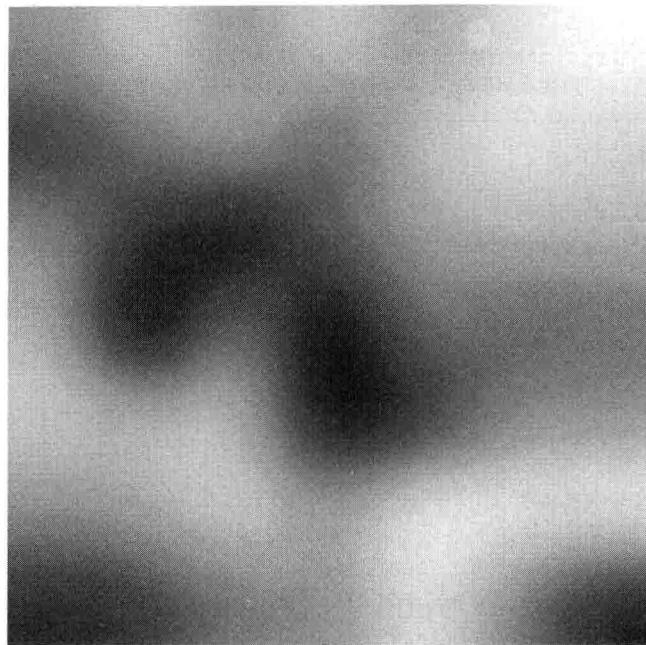


图 6.61 插值效果图（2）

效果很明显，第一张全是一个个的像素，非常影响美观。另外一张却有雾化的朦胧美感，所以插值方式的选择，对经过多次放大缩小的图片最终得到的效果是有很大影响的。

接着我们来看两种 resize 的调用范例。

### （1）方式一

调用范例：

```
Mat dstImage=Mat::zeros(512 ,512, CV_8UC3 );//新建一张 512x512 尺寸的图片  
Mat srcImage imread("1.jpg");  
//显式指定 dsize=dstImage.size(), 那么 fx 和 fy 会自动计算出来，不用额外指定。  
resize(srcImage, dstImage, dstImage.size());
```

### （2）方式二

调用范例：

```
Mat dstImage;  
Mat srcImage imread("1.jpg")  
//指定 fx 和 fy，让函数计算出目标图像的大小。  
resize(srcImage, dstImage, Size(), 0.5, 0.5);
```

接着我们看看完整的示例程序。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所依赖的头文件和命名空间  
-----  
#include <opencv2/opencv.hpp>  
#include <opencv2/imgproc/imgproc.hpp>  
using namespace cv;
```

```
-----【 main() 函数 】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
-----
int main()
{
    //载入原始图
    Mat srcImage = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材
图
    Mat tmpImage,dstImage1,dstImage2;//临时变量和目标图的定义
    tmpImage=srcImage;//将原始图赋给临时变量

    //显示原始图
    imshow("【 原始图 】", srcImage);

    //进行尺寸调整操作
    resize(tmpImage,dstImage1,Size(tmpImage.cols/2,
tmpImage.rows/2 ),(0,0),(0,0),3);
    resize(tmpImage,dstImage2,Size(tmpImage.cols*2,
tmpImage.rows*2 ),(0,0),(0,0),3);

    //显示效果图
    imshow("【 效果图 】之一", dstImage1);
    imshow("【 效果图 】之二", dstImage2);

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

程序运行截图如图 6.62 所示。



图 6.62 程序运行截图

## 6.6.6 图像金字塔相关 API 函数

图像金字塔相关 API 函数主要是 `pyrUp`、`pyrDown` 这一对，下面分别对其进行讲解。

### 1. 向上采样: pyrUp()函数

pyrUp()函数的作用是向上采样并模糊一张图像，说白了就是放大一张图片。

```
C++: void pyrUp(InputArray src, OutputArray dst, const Size&
dstsize=Size(), int borderType=BORDER_DEFAULT )
```

- 第一个参数, InputArray 类型的 src, 输入图像, 即源图像, 填 Mat 类的对象即可。
- 第二个参数, OutputArray 类型的 dst, 输出图像, 和源图片有一样的尺寸和类型。
- 第三个参数, const Size&类型的 dstsize, 输出图像的大小;有默认值 Size(), 即默认情况下, 由 Size (src.cols\*2, src.rows\*2) 来进行计算, 且一直需要满足下列条件:

$$| \text{dstsize.width} - \text{src.cols} * 2 | \leq (\text{dstsize.width} \bmod 2)$$

$$| \text{dstsize.height} - \text{src.rows} * 2 | \leq (\text{dstsize.height} \bmod 2)$$

- 第四个参数, int 类型的 borderType, 边界模式, 一般不用去管它。

pyrUp 函数执行高斯金字塔的采样操作, 其实它也可以用于拉普拉斯金字塔的。

首先, 它通过插入可为零的行与列, 对源图像进行向上取样操作, 然后将结果与 pyrDown()乘以 4 的内核做卷积。

下面直接看完整的示例程序。

```
-----【头文件、命名空间包含部分】-----
//      描述: 包含程序所依赖的头文件和命名空间
//-----  

#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace cv;  
  
-----【main()函数】-----
//      描述: 控制台应用程序的入口函数, 我们的程序从这里开始
//-----  

int main()
{
    //载入原始图
    Mat srcImage = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材
    图
    Mat tmpImage, dstImage;//临时变量和目标图的定义
    tmpImage=srcImage;//将原始图赋给临时变量

    //显示原始图
    imshow("【原始图】", srcImage);
    //进行向上取样操作
    pyrUp(tmpImage, dstImage, Size(tmpImage.cols*2,
    tmpImage.rows*2) );
    //显示效果图
```

```

imshow("【效果图】", dstImage);

waitKey(0);

return 0;
}

```

程序运行截图如图 6.63 和图 6.64 所示。



图 6.63 原始图



图 6.64 上取样效果图

## 2. 采样: pyrDown()函数

pyrDown()函数的作用是向下采样并模糊一张图片，说白了就是缩小一张图片。

```
C++: void pyrDown(InputArray src, OutputArray dst, const Size&
dstsize=Size(), int borderType=BORDER_DEFAULT)
```

- 第一个参数, InputArray 类型的 src, 输入图像, 即源图像, 填 Mat 类的对象即可。
- 第二个参数, OutputArray 类型的 dst, 输出图像, 和源图片有一样的尺寸和类型。
- 第三个参数, const Size&类型的 dstsize, 输出图像的大小;有默认值 Size(), 即默认情况下, 由 Size Size((src.cols+1)/2, (src.rows+1)/2)来进行计算, 且一直需要满足下列条件:

$$| \text{dstsize.width}^2 - \text{src.cols} | \leq 2$$

$$| \text{dstsize.height}^2 - \text{src.rows} | \leq 2$$

该 pyrDown 函数执行了高斯金字塔建造的向下采样的步骤。首先, 它将源图像与如下内核做卷积运算:

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

接着，它通过对图像的偶数行和列做插值来进行向下采样操作。

依然是看看完整的示例程序，如下。

```
-----【头文件、命名空间包含部分】-----
//    描述：包含程序所依赖的头文件和命名空间
//-----
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace cv;

-----【main() 函数】-----
//    描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----
int main()
{
    //载入原始图
    Mat srcImage = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材
    图
    Mat tmpImage,dstImage;//临时变量和目标图的定义
    tmpImage=srcImage;//将原始图赋给临时变量

    //显示原始图
    imshow("【原始图】", srcImage);
    //进行向下取样操作
    pyrDown( tmpImage, dstImage, Size( tmpImage.cols/2,
    tmpImage.rows/2 ) );
    //显示效果图
    imshow("【效果图】", dstImage);

    waitKey(0);

    return 0;
}
```

程序运行截图如图 6.65 和图 6.66 所示。



图 6.65 原始图



图 6.66 向下取样效果图

### 6.6.7 综合示例：图像金字塔与图片尺寸缩放

这个示例程序中，分别演示了用 resize, pyrUp, pyrDown 来让源图像进行放大缩小的操作，分别用键盘按键 1、2、3、4、A、D、W、S 来控制图片的放大与缩小。如图 6.67 所示。

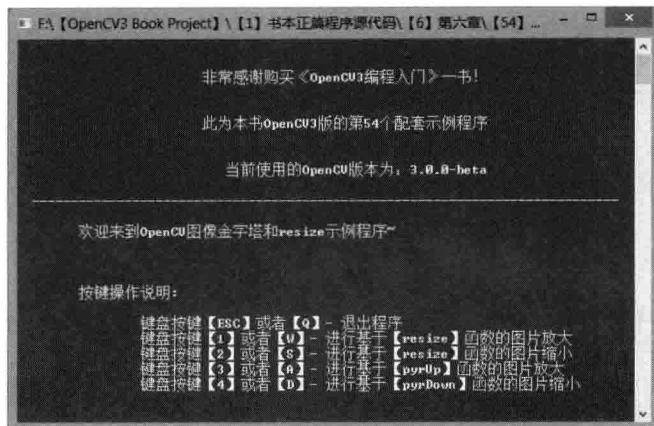


图 6.67 程序操作说明

详细注释的代码如下。

```

//-----【头文件、命名空间包含部分】-----
//    描述：包含程序所依赖的头文件和命名空间
//-----

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace cv;

//-----【宏定义部分】-
//    描述：定义一些辅助宏
//-

#define WINDOW_NAME "【程序窗口】"           //为窗口标题定义的宏

//-----【全局变量声明部分】-
//    描述：全局变量声明
//-

Mat g_srcImage, g_dstImage, g_tmpImage;

//-----【main() 函数】-
//    描述：控制台应用程序的入口函数，我们的程序从这里开始
//-

int main()
{

```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//载入原图
g_srcImage = imread("1.jpg"); //工程目录下需要有一张名为 1.jpg 的测试图像,
且其尺寸需被 2 的 N 次方整除, N 为可以缩放的次数
if( !g_srcImage.data ) { printf("读取 srcImage 错误~! \n"); return
false; }

// 创建显示窗口
namedWindow( WINDOW_NAME, WINDOW_AUTOSIZE );
imshow(WINDOW_NAME, g_srcImage);

//参数赋值
g_tmpImage = g_srcImage;
g_dstImage = g_tmpImage;

int key =0;

//轮询获取按键信息
while(1)
{
    key=waitKey(9) ; //读取键值到 key 变量中

    //根据 key 变量的值, 进行不同的操作
    switch(key)
    {
        //=====【 程序退出相关键值处理 】=====
        case 27://按键 ESC
            return 0;
            break;

        case 'q'://按键 Q
            return 0;
            break;

        //=====【 图片放大相关键值处理 】=====
        case 'a'://按键 A 按下, 调用 pyrUp 函数
            pyrUp( g_tmpImage, g_dstImage, Size( g_tmpImage.cols*2,
g_tmpImage.rows*2 ) );
            printf( ">检测到按键【A】被按下, 开始进行基于【pyrUp】函数的图片放
大: 图片尺寸×2 \n" );
            break;

        case 'w'://按键 W 按下, 调用 resize 函数
            resize(g_tmpImage,g_dstImage,Size( g_tmpImage.cols*2,
g_tmpImage.rows*2 ) );
            printf( ">检测到按键【W】被按下, 开始进行基于【resize】函数的图片放
大: 图片尺寸×2 \n" );
            break;

        case 'l'://按键 L 按下, 调用 resize 函数
            resize(g_tmpImage,g_dstImage,Size( g_tmpImage.cols*2,
g_tmpImage.rows*2 ) );
```

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

        printf( ">检测到按键【1】被按下，开始进行基于【resize】函数的图片放
大：图片尺寸×2 \n" );
        break;

    case '3': //按键 3 按下，调用 pyrUp 函数
        pyrUp( g_tmpImage, g_dstImage, Size( g_tmpImage.cols*2,
g_tmpImage.rows*2 ) );
        printf( ">检测到按键【3】被按下，开始进行基于【pyrUp】函数的图片放
大：图片尺寸×2 \n" );
        break;
//=====【图片缩小相关键值处理】=====
    case 'd': //按键 D 按下，调用 pyrDown 函数
        pyrDown( g_tmpImage, g_dstImage, Size( g_tmpImage.cols/2,
g_tmpImage.rows/2 ) );
        printf( ">检测到按键【D】被按下，开始进行基于【pyrDown】函数的图片
缩小：图片尺寸/2\n" );
        break;

    case 's' : //按键 S 按下，调用 resize 函数
        resize(g_tmpImage,g_dstImage,Size( g_tmpImage.cols/2,
g_tmpImage.rows/2 ) );
        printf( ">检测到按键【S】被按下，开始进行基于【resize】函数的图片缩
小：图片尺寸/2\n" );
        break;

    case '2'://按键 2 按下，调用 resize 函数
        resize(g_tmpImage,g_dstImage,Size( g_tmpImage.cols/2,
g_tmpImage.rows/2 ),(0,0),(0,0),2);
        printf( ">检测到按键【2】被按下，开始进行基于【resize】函数的图片缩
小：图片尺寸/2\n" );
        break;

    case '4': //按键 4 按下，调用 pyrDown 函数
        pyrDown( g_tmpImage, g_dstImage, Size( g_tmpImage.cols/2,
g_tmpImage.rows/2 ) );
        printf( ">检测到按键【4】被按下，开始进行基于【pyrDown】函数的图片
缩小：图片尺寸/2\n" );
        break;
    }

    //经过操作后，显示变化后的图
    imshow( WINDOW_NAME, g_dstImage );

    //将 g_dstImage 赋给 g_tmpImage，方便下一次循环
    g_tmpImage = g_dstImage;
}

return 0;
}

```

我们用键盘按键 1、2、3、4、A、D、W、S 来控制图片的放大与缩小，得到大小各异的图片窗口，并被记录在控制台窗口中。原图和效果图如图 6.68 和图 6.69

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

所示。



图 6.68 原始图

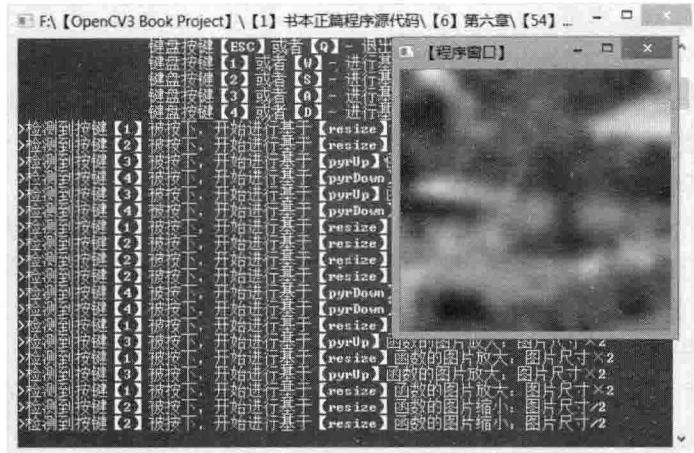


图 6.69 经过多次按键后的效果图

## 6.7 阈值化

在对各种图形进行处理操作的过程中，我们常常需要对图像中的像素做出取舍与决策，直接剔除一些低于或者高于一定值的像素。

阈值可以被视作最简单的图像分割方法。比如，从一副图像中利用阈值分割出我们需要的物体部分（当然这里的物体可以是一部分或者整体）。这样的图像分割方法基于图像中物体与背景之间的灰度差异，而且此分割属于像素级的分割。为了从一副图像中提取出我们需要的部分，应该用图像中的每一个像素点的灰度值与选取的阈值进行比较，并作出相应的判断。注意：阈值的选取依赖于具体的问题。即物体在不同的图像中有可能会有不同的灰度值。

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

一旦找到了需要分割的物体的像素点，可以对这些像素点设定一些特定的值来表示。例如，可以将该物体的像素点的灰度值设定为“0”（黑色），其他的像素点的灰度值为“255”（白色）。当然像素点的灰度值可以任意，但最好设定的两种颜色对比度较强，以方便观察结果。

在OpenCV 2.X中，Threshold()函数（基本阈值操作）和adaptiveThreshold()函数（自适应阈值操作）可以完成这样的要求。它们的基本思想是：给定一个数组和一个阈值，然后根据数组中的每个元素的值是高于还是低于阈值而进行一些处理。下面，我们将对这两个函数分别进行剖析。

### 6.7.1 固定阈值操作：Threshold()函数

函数 Threshold()对单通道数组应用固定阈值操作。该函数的典型应用是对灰度图像进行阈值操作得到二值图像，（compare()函数也可以达到此目的）或者是去掉噪声，例如过滤很小或很大象素值的图像点。

```
C++: double threshold(InputArray src, OutputArray dst, double thresh,
double maxval, int type)
```

- 第一个参数，InputArray类型的src，输入数组，填单通道，8或32位浮点类型的Mat即可。
- 第二个参数，OutputArray类型的dst，函数调用后的运算结果存在这里，即这个参数用于存放输出结果，且和第一个参数中的Mat变量有一样的尺寸和类型。
- 第三个参数，double类型的thresh，阈值的具体值。
- 第四个参数，double类型的maxval，当第五个参数阈值类型type取CV\_THRESH\_BINARY或CV\_THRESH\_BINARY\_INV时阈值类型时的最大值（对应地，OpenCV2中可以为CV\_THRESH\_BINARY和CV\_THRESH\_BINARY\_INV）。
- 第五个参数，int类型的type，阈值类型。threshold()函数支持的对图像取阈值的方法由其确定，具体用法如图6.70。

• THRESH_BINARY
$dst(x,y) = \begin{cases} maxval & \text{if } src(x,y) > thresh \\ 0 & \text{otherwise} \end{cases}$
• THRESH_BINARY_INV
$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > thresh \\ maxval & \text{otherwise} \end{cases}$
• THRESH_TRUNC
$dst(x,y) = \begin{cases} threshold & \text{if } src(x,y) > thresh \\ src(x,y) & \text{otherwise} \end{cases}$
• THRESH_TOZERO
$dst(x,y) = \begin{cases} src(x,y) & \text{if } src(x,y) > thresh \\ 0 & \text{otherwise} \end{cases}$
• THRESH_TOZERO_INV
$dst(x,y) = \begin{cases} 0 & \text{if } src(x,y) > thresh \\ src(x,y) & \text{otherwise} \end{cases}$

图6.70 Threshold()函数中阈值类型选项对应的操作

上述标识符依次取值分别为 0, 1, 2, 3, 4。

而针对上述公式，一一对应的图形化的阈值描述如图 6.71 所示（THRESH\_BINARY 对应二进制阈值，依次类推）。

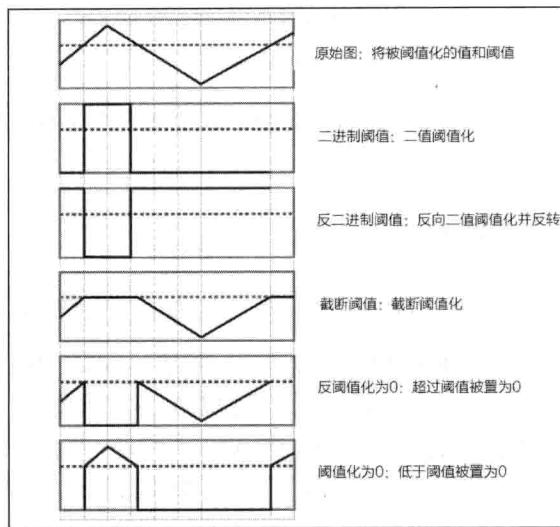


图 6.71 Threshold()函数中不同阈值类型的操作结果

## 6.7.2 自适应阈值操作: adaptiveThreshold()函数

adaptiveThreshold()函数的作用是对矩阵采用自适应阈值操作，支持就地操作。函数原型如下。

```
C++: void adaptiveThreshold(InputArray src, OutputArray dst, double maxValue, int adaptiveMethod, int thresholdType, int blockSize, double C)
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可，且需为 8 位单通道浮点型图像。
- 第二个参数，OutputArray 类型的 dst，函数调用后的运算结果存在这里，需和源图片有一样的尺寸和类型。
- 第三个参数，double 类型的 maxValue，给像素赋的满足条件的非零值。具体看下面的讲解。
- 第四个参数，int 类型的 adaptiveMethod，用于指定要使用的自适应阈值算法，可取值为 ADAPTIVE\_THRESH\_MEAN\_C 或 ADAPTIVE\_THRESH\_GAUSSIAN\_C。
- 第五个参数，int 类型的 thresholdType，阈值类型。取值必须为 THRESH\_BINARY、THRESH\_BINARY\_INV 其中之一。
- 第六个参数，int 类型的 blockSize，用于计算阈值大小的一个像素的邻域尺寸，取值为 3、5、7 等。
- 第七个参数，double 类型的 C，减去平均或加权平均值后的常数值。通常其为正数，但少数情况下也可以为零或负数。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

adaptiveThreshold()函数根据如下公式，将一幅灰度图变换为一幅二值图。

当第五个参数“阈值类型”thresholdType 取值为 THRESH\_BINARY 时，公式如下。

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

当第五个参数“阈值类型”thresholdType 取值为 THRESH\_BINARY\_INV 时，公式为：

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

而其中的  $T(x,y)$  为分别计算每个单独像素的阈值，取值如下。

- 对于 ADAPTIVE\_THRESH\_MEAN\_C 方法，阈值  $T(x,y)$  为  $blockSize \times blockSize$  邻域内  $(x, y)$  减去第七个参数 C 的平均值。
- 对于 ADAPTIVE\_THRESH\_GAUSSIAN\_C 方法，阈值  $T(x,y)$  为  $blockSize \times blockSize$  邻域内  $(x,y)$  减去第七个参数 C 与高斯窗交叉相关 (cross-correlation with a Gaussian window) 的加权总和。

### 6.7.3 示例程序：基本阈值操作

讲解完这个函数，让我们看一个调用示例程序，这个示例程序演示了基本阈值操作的方方面面。此程序可以通过按键，在不同的阈值模式之间切换，操作说明如图 6.72 所示。

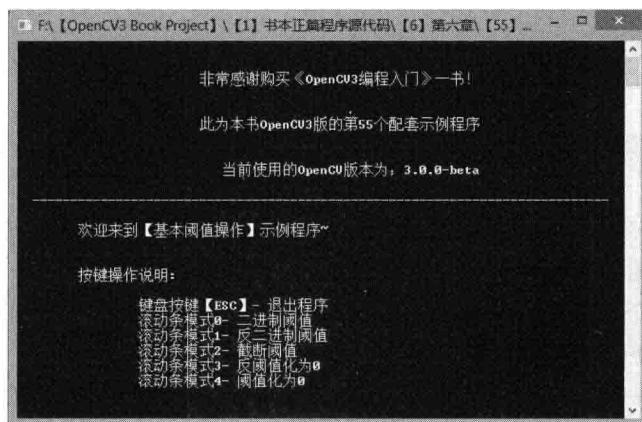


图 6.72 程序操作说明

其经过详细注释的示例程序如下。

```

-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----[头文件、命名空间包含部分]-----

#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
#include <iostream>
using namespace cv;
using namespace std;

//-----【宏定义部分】-----
//    描述：定义一些辅助宏
//-----
#define WINDOW_NAME "【程序窗口】"          //为窗口标题定义的宏

//-----【全局变量声明部分】-----
//    描述：全局变量的声明
//-----
int g_nThresholdValue = 100;
int g_nThresholdType = 3;
Mat g_srcImage, g_grayImage, g_dstImage;

//-----【全局函数声明部分】-----
//    描述：全局函数的声明
//-----
static void ShowHelpText(); //输出帮助文字
void on_Threshold( int, void* ); //回调函数

//-----【main()函数】-----
//    描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----
int main()
{
    //【1】读入源图片
    g_srcImage = imread("1.jpg");
    if(!g_srcImage.data) { printf("读取图片错误，请确定目录下是否有 imread\n函数指定的图片存在~!\n"); return false; }

    //【2】存留一份原图的灰度图
    cvtColor( g_srcImage, g_grayImage, COLOR_RGB2GRAY );

    //【3】创建窗口并显示原始图
    namedWindow( WINDOW_NAME, WINDOW_AUTOSIZE );

    //【4】创建滑动条来控制阈值
    createTrackbar( "模式",
                    WINDOW_NAME, &g_nThresholdType,
                    4, on_Threshold );

    createTrackbar( "参数值",
                    WINDOW_NAME, &g_nThresholdValue,
                    255, on_Threshold );

    //【5】初始化自定义的阈值回调函数
    on_Threshold( 0, 0 );
}
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
// 【6】轮询等待用户按键，如果 ESC 键按下则退出程序
while(1)
{
    int key;
    key = waitKey( 20 );
    if( (char)key == 27 ) { break; }
}

//-----【on_Threshold() 函数】-----
//      描述：自定义的阈值回调函数
//-----
void on_Threshold( int, void* )
{
    //调用阈值函数

threshold(g_grayImage,g_dstImage,g_nThresholdValue,255,g_nThresholdType);

    //更新效果图
    imshow( WINDOW_NAME, g_dstImage );
}
```

运行此程序，我们得到带有滑动条的窗口，通过调节滑动条，可以得到不同的阈值效果。原始图如图 6.73 所示，效果图如图 6.74~6.78 所示。



图 6.73 原始图



图 6.74 模式 0 (二进制阈值) 效果图

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

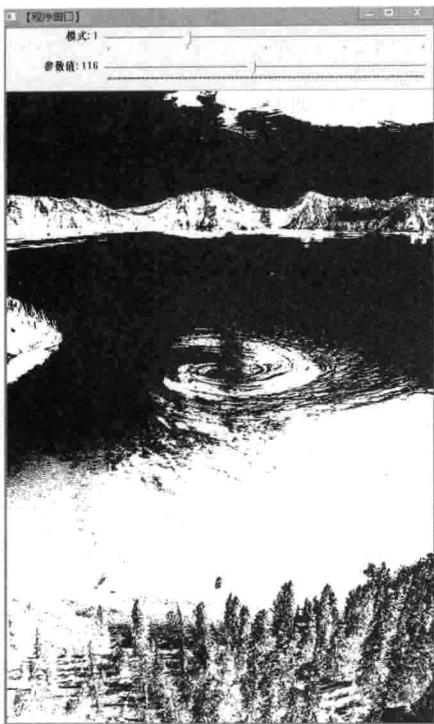


图 6.75 模式 1（反二进制阈值）效果图



图 6.76 模式 2（截断阈值）效果图

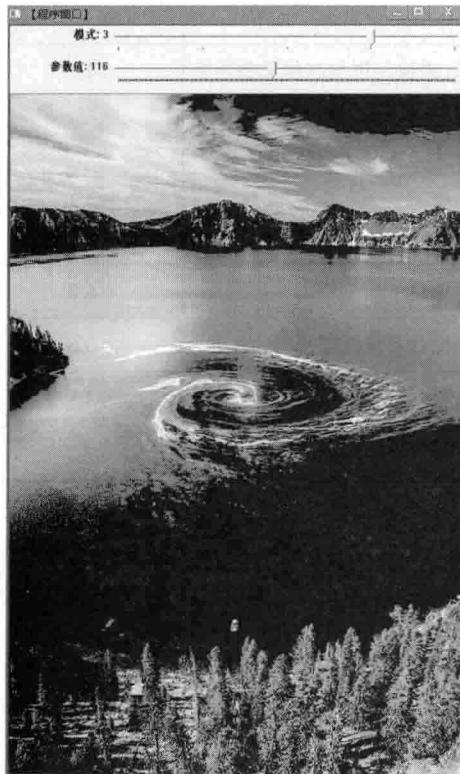


图 6.77 模式 3（反阈值化为 0）效果图

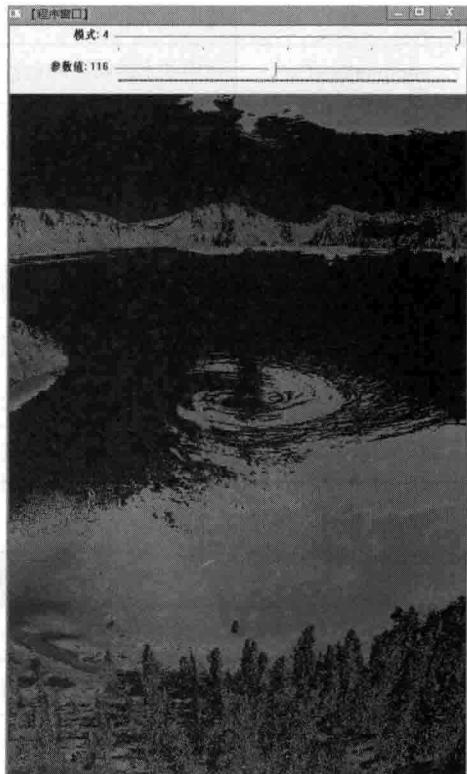


图 6.78 模式 4（阈值化为 0）效果图

## 6.8 本章小结

本章中我们学习了各种利用 OpenCV 进行图像处理的方法。包括属于线性滤波的方框滤波、均值滤波与高斯滤波，属于非线性滤波的中值滤波、双边滤波；两种基本形态学操作——膨胀与腐蚀；5 种高级形态学滤波操作——开运算、闭运算、形态学梯度、顶帽以及黑帽；还有漫水填充算法、图像金字塔、图像缩放、阈值化。涉及到的内容可谓非常丰富。相信学完此章的内容，你对 OpenCV 的了解已经上了一个层次。

### 本章核心函数清单

函数名称	说明	对应讲解章节
boxFilter	使用方框滤波来模糊一张图片	6.1.11
blur	对输入的图像进行均值滤波操作	6.1.11
GaussianBlur	用高斯滤波器来模糊一张图片	6.1.11
medianBlur	使用中值滤波器来模糊一张图片	6.2.4
bilateralFilter	用双边滤波器来模糊处理一张图片	6.2.4
dilate	使用像素邻域内的局部极大运算符来膨胀一张图片	6.3.5
erode	使用像素邻域内的局部极小运算符来腐蚀一张图片	6.3.5
morphologyEx	利用基本的膨胀和腐蚀技术，来执行更加高级形态学变换，如开闭运算、形态学梯度、顶帽、黑帽等，也可以实现最基本的图像膨胀和腐蚀	6.4.7
floodFill	用指定的颜色从种子点开始填充一个连接域，实现漫水填充算法	6.5.3
pyrUp	向上采样并模糊一张图片，说白了就是放大一张图片	6.6.6
pyrDown	向下采样并模糊一张图片，说白了就是缩小一张图片	6.6.6
Threshold	对单通道数组应用固定阈值操作	6.7.1
adaptiveThreshold	对矩阵采用自适应阈值操作	6.7.2

### 本章示例程序清单

示例程序序号	程序说明	对应章节
31	方框滤波：boxFilter 函数的使用	6.1.11
32	均值滤波：blur 函数的使用	6.1.11
33	高斯滤波：GaussianBlur 函数的使用	6.1.11
34	综合示例：图像线性滤波	6.1.12
35	中值滤波：medianBlur 函数的使用	6.2.4

续表

示例程序序号	程序说明	对应章节
36	双边滤波： bilateralFilter 函数的使用	6.2.4
37	综合示例： 图像滤波	6.2.5
38	膨胀： dilate 函数的使用	6.3.5
39	腐蚀： erode 函数的使用	6.3.5
40	综合示例： 腐蚀与膨胀	6.3.6
41	用 morphologyEx() 函数实现形态学膨胀	6.4.8
42	用 morphologyEx() 函数实现形态学腐蚀	6.4.8
43	用 morphologyEx() 函数实现形态学开运算	6.4.8
44	用 morphologyEx() 函数实现形态学闭运算	6.4.8
45	用 morphologyEx() 函数实现形态学梯度	6.4.8
46	用 morphologyEx() 函数实现形态学“顶帽”	6.4.8
47	用 morphologyEx() 函数实现形态学“黑帽”	6.4.8
48	综合示例： 形态学滤波	6.4.9
49	漫水填充算法： floodFill 函数	6.5.3
50	综合示例： 漫水填充	6.5.4
51	尺寸调整： resize() 函数的使用	6.6.5
52	向上采样图像金字塔： pyrUp() 函数的使用	6.6.6
53	向下采样图像金字塔： pyrDown() 函数的使用	6.6.6
54	综合示例： 图像金字塔与图片尺寸缩放	6.6.7
55	示例程序： 基本阈值操作	6.7.3

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

# 第 7 章

# 图像变换

## 导读

---

上一章中我们讲解了许多类型的图像处理方法。而迄今为止，所介绍的大多数操作都用于增强、修改或者“处理”图像，使之成为类似但全新的图像。

本章关注的重点是图像变换（image transform），即将一幅图像转变成图像数据的另一种表现形式。变换最常见的例子就是傅里叶变换（Fourier transform），即：将图像转换成源图像数据的另一种表示形式。这类操作的结果仍然保存为 OpenCV 图像结构的形式，但是新图像的每个单独像素表示原始输出图像的频谱分量，而不是通常所考虑的空间分量。

其实，计算机视觉中经常会用到许多有用的变换。OpenCV 提供了一套完整的实现工具和方法，可以帮助我们实现各种图像变换操作。

## 本章中，你将学到：

---

- 基于 OpenCV 的边缘检测
- 霍夫变换
- 重映射
- 仿射变换
- 直方图均衡化

## 7.1 基于 OpenCV 的边缘检测

本节中，我们将一起学习 OpenCV 中边缘检测的各种算子和滤波器——Canny 算子、Sobel 算子、Laplacian 算子以及 Scharr 滤波器。

### 7.1.1 边缘检测的一般步骤

在具体介绍之前，先来一起看看边缘检测的一般步骤。

#### 1. 【第一步】滤波

边缘检测的算法主要是基于图像强度的一阶和二阶导数，但导数通常对噪声很敏感，因此必须采用滤波器来改善与噪声有关的边缘检测器的性能。常见的滤波方法主要有高斯滤波，即采用离散化的高斯函数产生一组归一化的高斯核，然后基于高斯核函数对图像灰度矩阵的每一点进行加权求和。

#### 2. 【第二步】增强

增强边缘的基础是确定图像各点邻域强度的变化值。增强算法可以将图像灰度点邻域强度值有显著变化的点凸显出来。在具体编程实现时，可通过计算梯度幅值来确定。

#### 3. 【第三步】检测

经过增强的图像，往往邻域中有很多点的梯度值比较大，而在特定的应用中，这些点并不是要找的边缘点，所以应该采用某种方法来对这些点进行取舍。实际工程中，常用的方法是通过阈值化方法来检测。

另外，需要注意，下文中讲到的 Laplacian 算子、sobel 算子和 Scharr 算子都是带方向的，所以，示例中我们分别写了 X 方向、Y 方向和最终合成的效果图。

### 7.1.2 canny 算子

#### 1. canny 算子简介

Canny 边缘检测算子是 John F.Canny 于 1986 年开发出来的一个多级边缘检测算法。更为重要的是，Canny 创立了边缘检测计算理论（Computational theory of edge detection），解释了这项技术是如何工作的。Canny 边缘检测算法以 Canny 的名字命名，被很多人推崇为当今最优的边缘检测的算法。

其中，Canny 的目标是找到一个最优的边缘检测算法，让我们看一下最优边缘检测的三个主要评价标准。

- 低错误率：标识出尽可能多的实际边缘，同时尽可能地减少噪声产生的误报。
- 高定位性：标识出的边缘要与图像中的实际边缘尽可能接近。

- **最小响应：**图像中的边缘只能标识一次，并且可能存在的图像噪声不应标识为边缘。

为了满足这些要求，Canny 使用了变分法，这是一种寻找满足特定功能的函数的方法。最优检测用 4 个指数函数项的和表示，但是它非常近似于高斯函数的一阶导数。

## 2. Canny 边缘检测的步骤

### (1) 【第一步】消除噪声

一般情况下，使用高斯平滑滤波器卷积降噪。以下显示了一个  $\text{size} = 5$  的高斯内核示例：

$$K = \frac{1}{139} \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

### (2) 【第二步】计算梯度幅值和方向

此处，按照 Sobel 滤波器的步骤来操作。

#### ① 运用一对卷积阵列（分别作用于 x 和 y 方向）

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

#### ② 使用下列公式计算梯度幅值和方向

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

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

而梯度方向一般取这 4 个可能的角度之一——0 度，45 度，90 度，135 度。

### (3) 【第三步】非极大值抑制

这一步排除非边缘像素，仅仅保留了一些细线条（候选边缘）。

### (4) 【第四步】滞后阈值

这是最后一步，Canny 使用了滞后阈值，滞后阈值需要两个阈值（高阈值和低阈值）：

- ① 若某一像素位置的幅值超过高阈值，该像素被保留为边缘像素。
- ② 若某一像素位置的幅值小于低阈值，该像素被排除。
- ③ 若某一像素位置的幅值在两个阈值之间，该像素仅仅在连接到一个高于高阈值的像素时被保留。



对于 Canny 函数的使用，推荐的高低阈值比在 2:1 到 3:1 之间。

### 3. Canny 边缘检测：Canny()函数

Canny 函数利用 Canny 算子来进行图像的边缘检测操作。

```
C++: void Canny(InputArray image, OutputArray edges, double threshold1,  
double threshold2, int apertureSize=3, bool L2gradient=false )
```

- 第一个参数，InputArray 类型的 image，输入图像，即源图像，填 Mat 类的对象即可，且需为单通道 8 位图像。
- 第二个参数，OutputArray 类型的 edges，输出的边缘图，需要和源图片有一样的尺寸和类型。
- 第三个参数，double 类型的 threshold1，第一个滞后性阈值。
- 第四个参数，double 类型的 threshold2，第二个滞后性阈值。
- 第五个参数，int 类型的 apertureSize，表示应用 Sobel 算子的孔径大小，其有默认值 3。
- 第六个参数，bool 类型的 L2gradient，一个计算图像梯度幅值的标识，有默认值 false。

需要注意的是，这个函数阈值 1 和阈值 2 两者中较小的值用于边缘连接，而较大的值用来控制强边缘的初始段，推荐的高低阈值比在 2:1 到 3:1 之间。

调用示例如下。

```
//载入原始图  
Mat src = imread("l.jpg"); //工程目录下应该有一张名为 l.jpg 的素材图  
Canny(src, src, 3, 9, 3 );  
imshow("【效果图】Canny 边缘检测", src);
```

### 4. 示例程序：Canny 边缘检测

OpenCV 中调用 Canny 函数的实例代码如下。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所依赖的头文件和命名空间  
-----  
#include <opencv2/opencv.hpp>  
#include<opencv2/highgui/highgui.hpp>  
#include<opencv2/imgproc/imgproc.hpp>  
using namespace cv;  
  
-----【main()函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始  
-----  
int main()  
{  
    //载入原始图
```

## SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
Mat src = imread("l.jpg"); //工程目录下应该有一张名为 l.jpg 的素材图
Mat src1=src.clone();

//显示原始图
imshow("【原始图】Canny 边缘检测", src);

//-----
// 一、最简单的 canny 用法，拿到原图后直接用。
// 注意：此方法在 OpenCV2 中可用，在 OpenCV3 中已失效
//-----
//Canny( srcImage, srcImage, 150, 100,3 );
//imshow("【效果图】Canny 边缘检测", srcImage);

//-----
// 二、高阶的 canny 用法，转成灰度图，降噪，用 canny，最后将得到的边缘作为掩
// 码，拷贝原图到效果图上，得到彩色的边缘图

//-----
Mat dst,edge,gray;

// 【1】创建与 src 同类型和大小的矩阵(dst)
dst.create( src1.size(), src1.type() );

// 【2】将原图像转换为灰度图像
cvtColor( src1, gray, COLOR_BGR2GRAY );

// 【3】先用使用 3x3 内核来降噪
blur( gray, edge, Size(3,3) );

// 【4】运行 Canny 算子
Canny( edge, edge, 3, 9,3 );

// 【5】将 g_dstImage 内的所有元素设置为 0
dst = Scalar::all(0);

// 【6】使用 Canny 算子输出的边缘图 g_cannyDetectedEdges 作为掩码，来将原图
g_srcImage 拷到目标图 g_dstImage 中
src1.copyTo( dst, edge);

// 【7】显示效果图
imshow("【效果图】Canny 边缘检测 2", dst);

waitKey(0);

return 0;
}
```

原始图和运行效果图如图 7.1、7.2、7.3 所示。

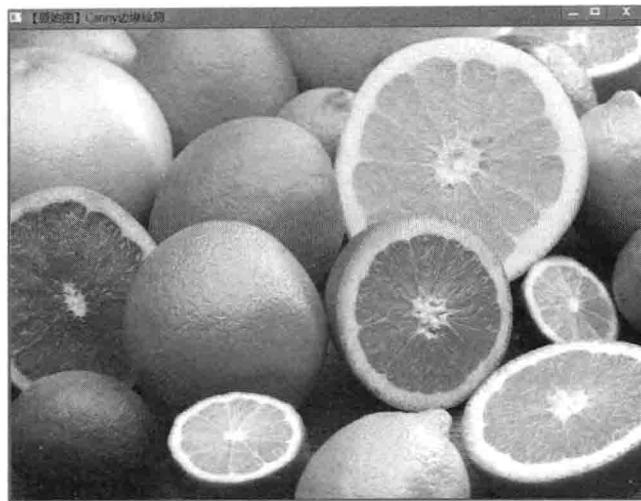


图 7.1 原始图

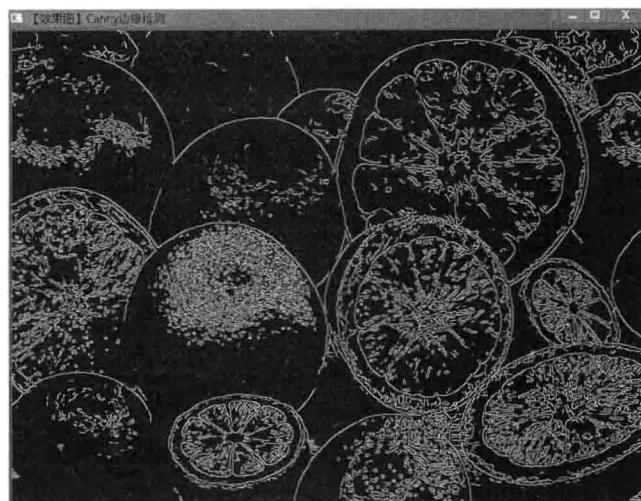


图 7.2 灰度 canny 检测图

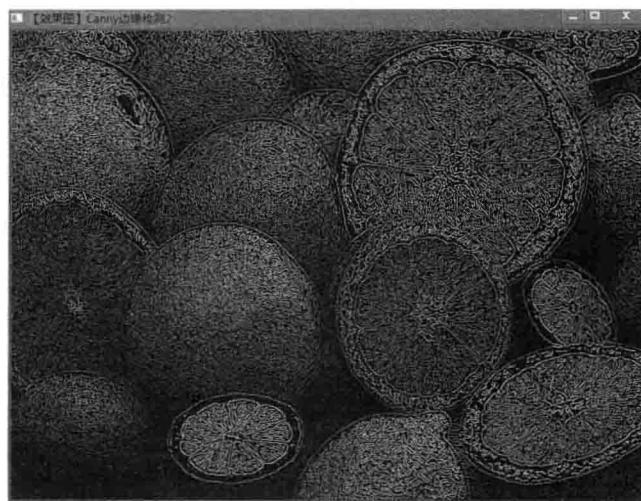


图 7.3 彩色 canny 检测图

### 7.1.3 sobel 算子

#### 1. sobel 算子的基本概念

Sobel 算子是一个主要用于边缘检测的离散微分算子（discrete differentiation operator）。它结合了高斯平滑和微分求导，用来计算图像灰度函数的近似梯度。在图像的任何一点使用此算子，都将会产生对应的梯度矢量或是其法矢量。

#### 2. sobel 算子的计算过程

我们假设被作用图像为 I 然后进行如下操作。

(1) 分别在 x 和 y 两个方向求导。

① 水平变化：将 I 与一个奇数大小的内核  $G_x$  进行卷积。比如，当内核大小为 3 时， $G_x$  的计算结果为：

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

② 垂直变化：将 I 与一个奇数大小的内核进行卷积。比如，当内核大小为 3 时，计算结果为：

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

(2) 在图像的每一点，结合以上两个结果求出近似梯度：

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

另外有时，也可用下面更简单的公式代替：

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

#### 3. 使用 Sobel 算子：Sobel()函数

Sobel 函数使用扩展的 Sobel 算子，来计算一阶、二阶、三阶或混合图像差分。

```
C++: void Sobel (
    InputArray src,
    OutputArray dst,
    int ddepth,
    int dx,
    int dy,
    int ksize=3,
    double scale=1,
    double delta=0,
    int borderType=BORDER_DEFAULT );
```

(1) 第一个参数，InputArray 类型的 src，为输入图像，填 Mat 类型即可。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

(2) 第二个参数, `OutputArray` 类型的 `dst`, 即目标图像, 函数的输出参数, 需要和源图片有一样的尺寸和类型。

(3) 第三个参数, `int` 类型的 `ddepth`, 输出图像的深度, 支持如下 `src.depth()` 和 `ddepth` 的组合:

- 若 `src.depth() = CV_8U`, 取 `ddepth = -1/CV_16S/CV_32F/CV_64F`
- 若 `src.depth() = CV_16U/CV_16S`, 取 `ddepth = -1/CV_32F/CV_64F`
- 若 `src.depth() = CV_32F`, 取 `ddepth = -1/CV_32F/CV_64F`
- 若 `src.depth() = CV_64F`, 取 `ddepth = -1/CV_64F`

(4) 第四个参数, `int` 类型 `dx`, `x` 方向上的差分阶数。

(5) 第五个参数, `int` 类型 `dy`, `y` 方向上的差分阶数。

(6) 第六个参数, `int` 类型 `ksize`, 有默认值 3, 表示 Sobel 核的大小; 必须取 1、3、5 或 7。

(7) 第七个参数, `double` 类型的 `scale`, 计算导数值时可选的缩放因子, 默认值是 1, 表示默认情况下是没有应用缩放的。可以在文档中查阅 `getDerivKernels` 的相关介绍, 来得到这个参数的更多信息。

(8) 第八个参数, `double` 类型的 `delta`, 表示在结果存入目标图 (第二个参数 `dst`) 之前可选的 `delta` 值, 有默认值 0。

(9) 第九个参数, `int` 类型的 `borderType`, 边界模式, 默认值为 `BORDER_DEFAULT`。这个参数可以在官方文档中 `borderInterpolate` 处得到更详细的信息。

一般情况下, 都是用  $\text{ksize} \times \text{ksize}$  内核来计算导数的。然而, 有一种特殊情况——当 `ksize` 为 1 时, 往往会使用  $3 \times 1$  或者  $1 \times 3$  的内核。且这种情况下, 并没有进行高斯平滑操作。

一些补充说明如下。

① 当内核大小为 3 时, Sobel 内核可能产生比较明显的误差 (毕竟, Sobel 算子只是求取了导数的近似值而已)。为解决这一问题, OpenCV 提供了 Scharr 函数, 但该函数仅作用于大小为 3 的内核。该函数的运算与 Sobel 函数一样快, 但结果却更加精确, 其内核是这样的:

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

② 因为 Sobel 算子结合了高斯平滑和分化 (differentiation), 因此结果会具有更多的抗噪性。大多数情况下, 我们使用 `sobel` 函数时, 取 `【xorder = 1, yorder = 0, ksize = 3】` 来计算图像 X 方向的导数, `【xorder = 0, yorder = 1, ksize = 3】` 来计算图像 y 方向的导数。

计算图像 X 方向的导数, 取 `【xorder=1, yorder=0, ksize=3】` 情况对应的内核:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 3 \end{bmatrix}$$

而计算图像 Y 方向的导数，取【xorder= 0, yorder = 1, ksize = 3】对应的内核：

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

#### 4. 示例程序：Sobel 算子的使用

调用 Sobel 函数的实例代码如下。这里只是教大家如何使用 Sobel 函数，就没有先用一句 cvtColor 将原图转化为灰度图，而是直接用彩色图操作。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----
#include <opencv2/opencv.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
using namespace cv;

-----【main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
-----
int main()
{
    //【0】创建 grad_x 和 grad_y 矩阵
    Mat grad_x, grad_y;
    Mat abs_grad_x, abs_grad_y, dst;

    //【1】载入原始图
    Mat src = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材图

    //【2】显示原始图
    imshow("【原始图】sobel 边缘检测", src);

    //【3】求 X 方向梯度
    Sobel( src, grad_x, CV_16S, 1, 0, 3, 1, 1, BORDER_DEFAULT );
    convertScaleAbs( grad_x, abs_grad_x );
    imshow("【效果图】X 方向 Sobel", abs_grad_x);

    //【4】求 Y 方向梯度
    Sobel( src, grad_y, CV_16S, 0, 1, 3, 1, 1, BORDER_DEFAULT );
    convertScaleAbs( grad_y, abs_grad_y );
    imshow("【效果图】Y 方向 Sobel", abs_grad_y);

    //【5】合并梯度(近似)
    addWeighted( abs_grad_x, 0.5, abs_grad_y, 0.5, 0, dst );
}
```

```

imshow("【效果图】整体方向 Sobel", dst);

waitKey(0);
return 0;
}

```

运行截图如图 7.4 和图 7.5 所示。

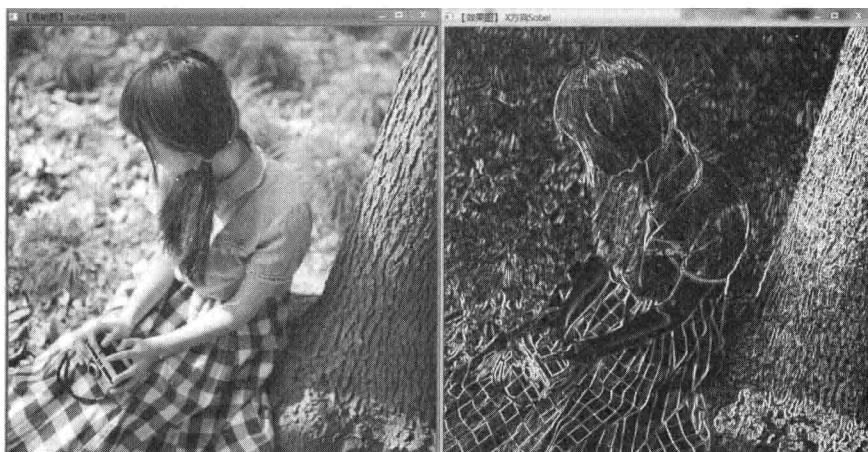


图 7.4 原始图和 X 方向上的 sobel

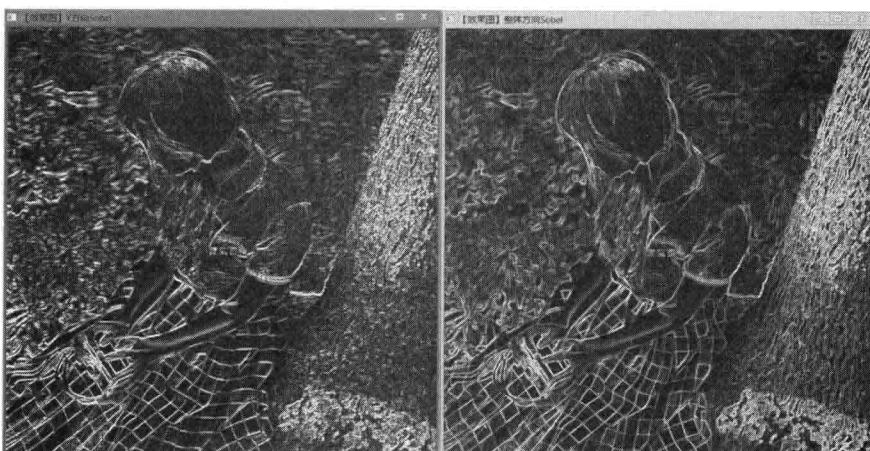


图 7.5 Y 方向上的 sobel 和整体方向上的 sobel

#### 7.1.4 Laplacian 算子

##### 1. Laplacian 算子简介

Laplacian 算子是  $n$  维欧几里德空间中的一个二阶微分算子，定义为梯度  $\text{grad}$  的散度  $\text{div}$ 。因此如果  $f$  是二阶可微的实函数，则  $f$  的拉普拉斯算子定义如下。

- (1)  $f$  的拉普拉斯算子也是笛卡儿坐标系  $x_i$  中的所有非混合二阶偏导数求和。
- (2) 作为一个二阶微分算子，拉普拉斯算子把  $C$  函数映射到  $C$  函数。对于  $k \geq 2$ ，表达式(1)（或(2)）定义了一个算子  $\Delta: C(R) \rightarrow C(R)$ ；或更一般地，对于任

何开集  $\Omega$ , 定义了一个算子  $\Delta: C(\Omega) \rightarrow C(\Omega)$ 。

根据图像处理的原理可知, 二阶导数可以用来进行检测边缘。因为图像是“二维”, 需要在两个方向进行求导。使用 Laplacian 算子将会使求导过程变得简单。

Laplacian 算子的定义:

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

需要说明的是, 由于 Laplacian 使用了图像梯度, 它内部的代码其实是调用了 Sobel 算子的。



让一幅图像减去它的 Laplacian 算子可以增强对比度。

## 2. 计算拉普拉斯变换: Laplacian()函数

Laplacian 函数可以计算出图像经过拉普拉斯变换后的结果。

```
C++: void Laplacian(InputArray src, OutputArray dst, int ddepth, int
ksize=1, double scale=1, double delta=0,
int borderType=BORDER_DEFAULT );
```

- 第一个参数, InputArray 类型的 image, 输入图像, 即源图像, 填 Mat 类的对象即可, 且需为单通道 8 位图像。
- 第二个参数, OutputArray 类型的 edges, 输出的边缘图, 需要和源图片有一样的尺寸和通道数。
- 第三个参数, int 类型的 ddept, 目标图像的深度。
- 第四个参数, int 类型的 ksize, 用于计算二阶导数的滤波器的孔径尺寸, 大小必须为正奇数, 且有默认值 1。
- 第五个参数, double 类型的 scale, 计算拉普拉斯值的时候可选的比例因子, 有默认值 1。
- 第六个参数, double 类型的 delta, 表示在结果存入目标图(第二个参数 dst)之前可选的 delta 值, 有默认值 0。
- 第七个参数, int 类型的 borderType, 边界模式, 默认值为 BORDER\_DEFAULT。这个参数可以在官方文档中 borderInterpolate() 处得到更详细的信息。

Laplacian()函数其实主要是利用 sobel 算子的运算。它通过加上 sobel 算子运算出的图像 x 方向和 y 方向上的导数, 来得到载入图像的拉普拉斯变换结果。

其中, sobel 算子 (ksize>1) 如下:

$$dst = \Delta src = \frac{\partial^2 src}{\partial x^2} + \frac{\partial^2 src}{\partial y^2}$$

而当 ksize=1 时, Laplacian()函数采用以下 3x3 的孔径:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

### 3. 示例程序：Laplacian 算子的使用

让我们看一看调用实例。

```

-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----

#include <opencv2/opencv.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
using namespace cv;

-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
-----

int main()
{
    //【0】变量的定义
    Mat src,src_gray,dst, abs_dst;

    //【1】载入原始图
    src = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材图

    //【2】显示原始图
    imshow("【原始图】图像 Laplace 变换", src);

    //【3】使用高斯滤波消除噪声
    GaussianBlur( src, src, Size(3,3), 0, 0, BORDER_DEFAULT );

    //【4】转换为灰度图
    cvtColor( src, src_gray, COLOR_RGB2GRAY );

    //【5】使用 Laplace 函数
    Laplacian( src_gray, dst, CV_16S, 3, 1, 0, BORDER_DEFAULT );

    //【6】计算绝对值，并将结果转换成 8 位
    convertScaleAbs( dst, abs_dst );

    //【7】显示效果图
    imshow( "【效果图】图像 Laplace 变换", abs_dst );

    waitKey(0);

    return 0;
}

```

运行截图如图 7.6 所示。



图 7.6 示例效果图

### 7.1.5 scharr 滤波器

我们一般直接称 scharr 为滤波器，而不是算子。上文已经讲到，它在 OpenCV 中主要是配合 Sobel 算子的运算而存在的。下面让我们直接来看看其函数讲解。

#### 1. 计算图像差分：Scharr() 函数

使用 Scharr 滤波器运算符计算 x 或 y 方向的图像差分。其实它的参数变量和 Sobel 基本上是一样的，除了没有 ksize 核的大小。

```
C++: void Scharr(
InputArray src, //源图
OutputArray dst, //目标图
int ddepth, //图像深度
int dx, // x 方向上的差分阶数
int dy, //y 方向上的差分阶数
double scale=1, //缩放因子
double delta=0, // delta 值
int borderType=BORDER_DEFAULT ) // 边界模式
```

- (1) 第一个参数，InputArray 类型的 src，为输入图像，填 Mat 类型即可。
- (2) 第二个参数，OutputArray 类型的 dst，即目标图像，函数的输出参数，需要和源图片有一样的尺寸和类型。
- (3) 第三个参数，int 类型的 ddepth，输出图像的深度，支持如下 src.depth() 和 ddepth 的组合：

- 若 src.depth() = CV\_8U，取 ddepth = -1/CV\_16S/CV\_32F/CV\_64F
- 若 src.depth() = CV\_16U/CV\_16S，取 ddepth = -1/CV\_32F/CV\_64F

- 若 `src.depth() = CV_32F`, 取 `ddepth = -1/CV_32F/CV_64F`
- 若 `src.depth() = CV_64F`, 取 `ddepth = -1/CV_64F`

(4) 第四个参数, int 类型 `dx`, `x` 方向上的差分阶数。

(5) 第五个参数, int 类型 `dy`, `y` 方向上的差分阶数。

(6) 第六个参数, double 类型的 `scale`, 计算导数值时可选的缩放因子, 默认值是 1, 表示默认情况下是没有应用缩放的。我们可以在文档中查阅 `getDerivKernels` 的相关介绍, 来得到这个参数的更多信息。

(7) 第七个参数, double 类型的 `delta`, 表示在结果存入目标图 (第二个参数 `dst`) 之前可选的 `delta` 值, 有默认值 0。

(8) 第八个参数, int 类型的 `borderType`, 边界模式, 默认值为 `BORDER_DEFAULT`。这个参数可以在官方文档中 `borderInterpolate` 处得到更详细的信息。

不难理解, 如下两者是等价的, 即:

```
Scharr(src, dst, ddepth, dx, dy, scale, delta, borderType);
```

与

```
Sobel(src, dst, ddepth, dx, dy, CV_SCHARR, scale, delta, borderType);
```

## 2. 示例程序: Scharr 滤波器

```
-----【头文件、命名空间包含部分】-----
//      描述: 包含程序所依赖的头文件和命名空间
//-----  

#include <opencv2/opencv.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
using namespace cv;  

-----【main()函数】-----
//      描述: 控制台应用程序的入口函数, 我们的程序从这里开始
//-----  

int main()
{
    //【0】创建 grad_x 和 grad_y 矩阵
    Mat grad_x, grad_y;
    Mat abs_grad_x, abs_grad_y,dst;

    //【1】载入原始图
    Mat src = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材图

    //【2】显示原始图
    imshow("【原始图】Scharr 滤波器", src);

    //【3】求 X 方向梯度
    Scharr( src, grad_x, CV_16S, 1, 0, 1, 0, BORDER_DEFAULT );
    convertScaleAbs( grad_x, abs_grad_x );
    imshow("【效果图】X 方向 Scharr", abs_grad_x);
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//【4】求Y方向梯度  
Scharr( src, grad_y, CV_16S, 0, 1, 1, 0, BORDER_DEFAULT );  
convertScaleAbs( grad_y, abs_grad_y );  
imshow("【效果图】Y方向 Scharr", abs_grad_y );  
  
//【5】合并梯度(近似)  
addWeighted( abs_grad_x, 0.5, abs_grad_y, 0.5, 0, dst );  
  
//【6】显示效果图  
imshow("【效果图】合并梯度后 Scharr", dst );  
  
waitKey(0);  
return 0;  
}
```

原图如图 7.7 所示，运行效果图如图 7.8、图 7.9、图 7.10 所示。



图 7.7 原始图

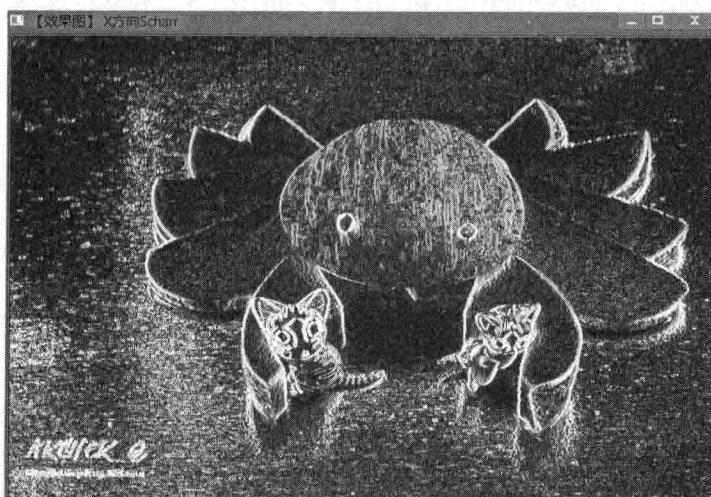


图 7.8 X 方向上的 Scharr

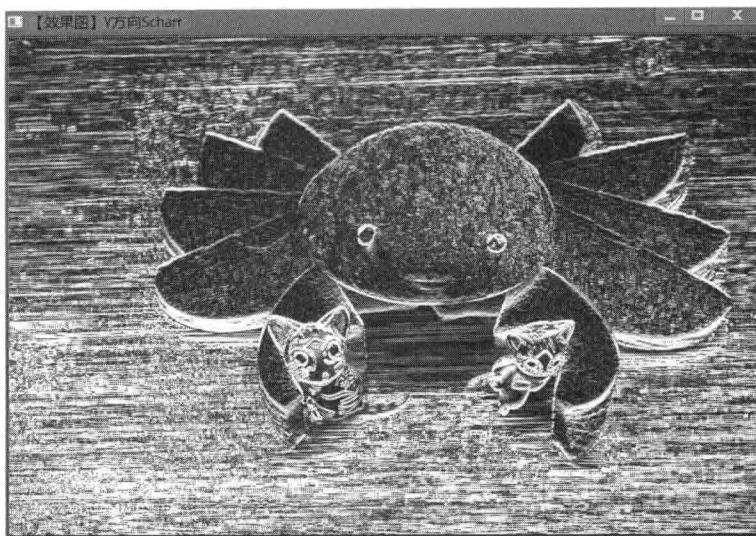


图 7.9 Y 方向上的 Scharr

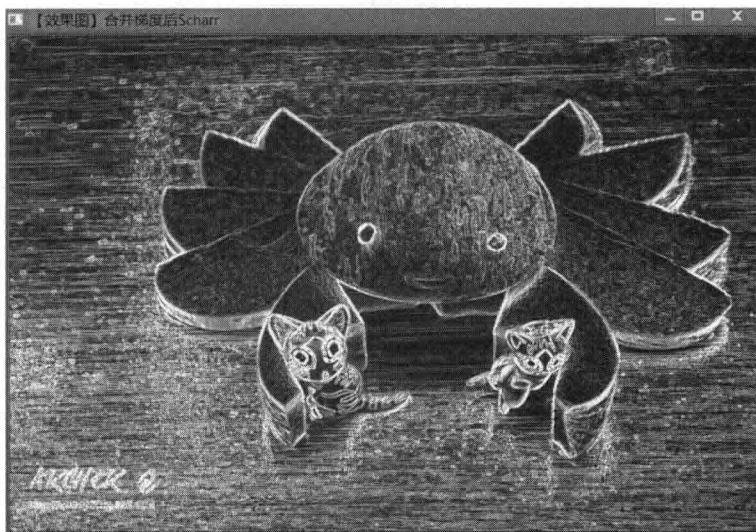


图 7.10 合并梯度后的 Scharr

### 7.1.6 综合示例：边缘检测

本节依然是配给大家一个详细注释的配套示例程序，把这节中介绍的知识点以代码为载体，更形象地展现出来。

这个示例程序中，分别演示了 canny 边缘检测、sobel 边缘检测、scharr 滤波器的使用，经过详细注释的的代码如下。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所依赖的头文件和命名空间  
-----  
#include <opencv2/highgui/highgui.hpp>  
#include <opencv2/imgproc/imgproc.hpp>  
using namespace cv;
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//-----【全局变量声明部分】-----  
//      描述：全局变量声明  
//-----  
//原图，原图的灰度版，目标图  
Mat g_srcImage, g_srcGrayImage, g_dstImage;  
  
//Canny 边缘检测相关变量  
Mat g_cannyDetectedEdges;  
int g_cannyLowThreshold=1;//TrackBar 位置参数  
  
//Sobel 边缘检测相关变量  
Mat g_sobelGradient_X, g_sobelGradient_Y;  
Mat g_sobelAbsGradient_X, g_sobelAbsGradient_Y;  
int g_sobelKernelSize=1;//TrackBar 位置参数  
  
//Scharr 滤波器相关变量  
Mat g_scharrGradient_X, g_scharrGradient_Y;  
Mat g_scharrAbsGradient_X, g_scharrAbsGradient_Y;  
  
//-----【全局函数声明部分】-----  
//      描述：全局函数声明  
//-----  
static void on_Canny(int, void*);//Canny 边缘检测窗口滚动条的回调函数  
static void on_Sobel(int, void*);//Sobel 边缘检测窗口滚动条的回调函数  
void Scharr(); //封装了 Scharr 边缘检测相关代码的函数  
  
//-----【main() 函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始  
//-----  
int main( int argc, char** argv )  
{  
    //改变 console 字体颜色  
    system("color 2F");  
  
    //载入原图  
    g_srcImage = imread("1.jpg");  
    if( !g_srcImage.data ) { printf("读取 srcImage 错误~! \n"); return  
false; }  
  
    //显示原始图  
    namedWindow("【原始图】");  
    imshow("【原始图】", g_srcImage);  
  
    // 创建与 src 同类型和大小的矩阵(dst)  
    g_dstImage.create( g_srcImage.size(), g_srcImage.type() );  
  
    // 将原图像转换为灰度图像  
    cvtColor( g_srcImage, g_srcGrayImage, COLOR_BGR2GRAY );
```

```
// 创建显示窗口
namedWindow( "【效果图】Canny 边缘检测", WINDOW_AUTOSIZE );
namedWindow( "【效果图】Sobel 边缘检测", WINDOW_AUTOSIZE );

// 创建 trackbar
createTrackbar( "参数值:", "【效果图】Canny 边缘检测",
&g_cannyLowThreshold, 120, on_Canny );
createTrackbar("参数值:", "【效果图】Sobel 边缘检测", &g_sobelKernelSize,
3, on_Sobel );

// 调用回调函数
on_Canny(0, 0);
on_Sobel(0, 0);

// 调用封装了 Scharr 边缘检测代码的函数
Scharr();

// 轮询获取按键信息，若按下 Q，程序退出
while((char(waitKey(1)) != 'q')) {}

return 0;
}

//-----【on_Canny() 函数】-----
//    描述：Canny 边缘检测窗口滚动条的回调函数
//-----
void on_Canny(int, void*)
{
    // 先使用 3x3 内核来降噪
    blur( g_srcGrayImage, g_cannyDetectedEdges, Size(3,3) );

    // 运行我们的 Canny 算子
    Canny( g_cannyDetectedEdges, g_cannyDetectedEdges,
g_cannyLowThreshold, g_cannyLowThreshold*3, 3 );

    // 先将 g_dstImage 内的所有元素设置为 0
    g_dstImage = Scalar::all(0);

    // 使用 Canny 算子输出的边缘图 g_cannyDetectedEdges 作为掩码，来将原图
    g_srcImage 拷贝到目标图 g_dstImage 中
    g_srcImage.copyTo( g_dstImage, g_cannyDetectedEdges);

    // 显示效果图
    imshow( "【效果图】Canny 边缘检测", g_dstImage );
}

//-----【on_Sobel() 函数】-----
//    描述：Sobel 边缘检测窗口滚动条的回调函数
//-----
void on_Sobel(int, void*)
{
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
// 求 X 方向梯度
Sobel( g_srcImage, g_sobelGradient_X, CV_16S, 1, 0,
(2*g_sobelKernelSize+1), 1, 1, BORDER_DEFAULT );
convertScaleAbs( g_sobelGradient_X, g_sobelAbsGradient_X );//计算
绝对值，并将结果转换成 8 位

// 求 Y 方向梯度
Sobel( g_srcImage, g_sobelGradient_Y, CV_16S, 0, 1,
(2*g_sobelKernelSize+1), 1, 1, BORDER_DEFAULT );
convertScaleAbs( g_sobelGradient_Y, g_sobelAbsGradient_Y );//计算
绝对值，并将结果转换成 8 位

// 合并梯度
addWeighted( g_sobelAbsGradient_X, 0.5, g_sobelAbsGradient_Y, 0.5,
0, g_dstImage );

// 显示效果图
imshow("【效果图】Sobel 边缘检测", g_dstImage);

}

//-----【 Scharr() 函数 】-----
//      描述：封装了 Scharr 边缘检测相关代码的函数
//-----
void Scharr()
{
    // 求 X 方向梯度
    Scharr( g_srcImage, g_scharrGradient_X, CV_16S, 1, 0, 1, 0,
BORDER_DEFAULT );
    convertScaleAbs( g_scharrGradient_X, g_scharrAbsGradient_X );//计
算绝对值，并将结果转换成 8 位

    // 求 Y 方向梯度
    Scharr( g_srcImage, g_scharrGradient_Y, CV_16S, 0, 1, 1, 0,
BORDER_DEFAULT );
    convertScaleAbs( g_scharrGradient_Y, g_scharrAbsGradient_Y );//计
算绝对值，并将结果转换成 8 位

    // 合并梯度
    addWeighted( g_scharrAbsGradient_X, 0.5, g_scharrAbsGradient_Y, 0.5,
0, g_dstImage );

    // 显示效果图
    imshow("【效果图】Scharr 滤波器", g_dstImage);
}
```

下面放出一些程序运行效果图。如图 7.11~7.15 所示。



图 7.11 原始图



图 7.12 canny 边缘检测效果图（1）

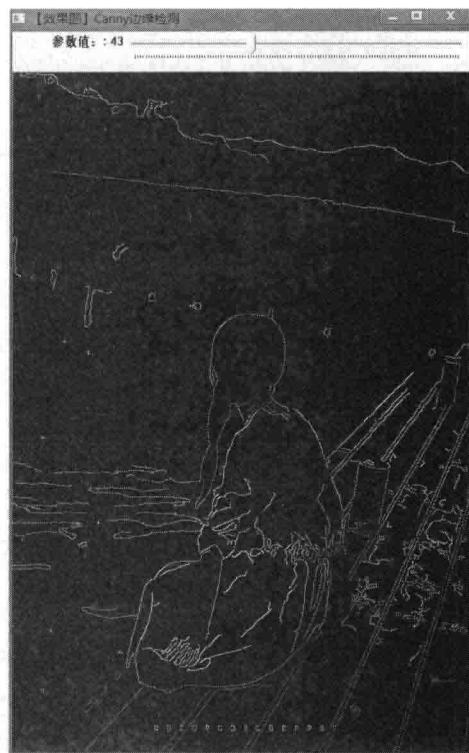


图 7.13 canny 边缘检测效果图（2）

## 7.2 霍夫变换

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>



图 7.14 Sobel 边缘检测效果图



图 7.15 Scharr 滤波器效果图

## 7.2 霍夫变换

本节中，我们将一起探讨 OpenCV 中霍夫变换相关的知识点，并了解了 OpenCV 中实现霍夫线变换的 HoughLines、HoughLinesP 函数的使用方法，以及实现霍夫圆变换的 HoughCircles 函数的使用方法。

在图像处理和计算机视觉领域中，如何从当前的图像中提取所需要的特征信息是图像识别的关键所在。在许多应用场合中需要快速准确地检测出直线或者圆。其中一种非常有效的解决问题的方法是霍夫（Hough）变换，其为图像处理中从图像中识别几何形状的基本方法之一，应用很广泛，也有很多改进算法。最基本的霍夫变换是从黑白图像中检测直线（线段）。本节就将介绍 OpenCV 中霍夫变换的使用方法和相关知识。

### 7.2.1 霍夫变换概述

霍夫变换（Hough Transform）是图像处理中的一种特征提取技术，该过程在一个参数空间中通过计算累计结果的局部最大值得到一个符合该特定形状的集合作为霍夫变换结果。霍夫变换于 1962 年由 Paul Hough 首次提出，最初的 Hough 变换是设计用来检测直线和曲线的。起初的方法要求知道物体边界线的解析方程，但不需要有关区域位置的先验知识。这种方法的一个突出优点是分割结果的 Robustness，即对数据的不完全或噪声不是非常敏感。然而，要获得描述边界的解

析表达常常是不可能的。后于 1972 年由 Richard Duda & Peter Hart 推广使用，经典霍夫变换用来检测图像中的直线，后来霍夫变换扩展到任意形状物体的识别，多为圆和椭圆。霍夫变换运用两个坐标空间之间的变换将在一个空间中具有相同形状的曲线或直线映射到另一个坐标空间的一个点上形成峰值，从而把检测任意形状的问题转化为统计峰值问题。

霍夫变换在 OpenCV 中分为霍夫线变换和霍夫圆变换两种，下面将分别进行介绍。

## 7.2.2 OpenCV 中的霍夫线变换

我们知道，霍夫线变换是一种用来寻找直线的方法。在使用霍夫线变换之前，首先要对图像进行边缘检测的处理，即霍夫线变换的直接输入只能是边缘二值图像。

OpenCV 支持三种不同的霍夫线变换，它们分别是：标准霍夫变换（Standard Hough Transform, SHT）、多尺度霍夫变换（Multi-Scale Hough Transform, MSHT）和累计概率霍夫变换（Progressive Probabilistic Hough Transform, PPHT）。

其中，多尺度霍夫变换（MSHT）为经典霍夫变换（SHT）在多尺度下的一个变种。而累计概率霍夫变换（PPHT）算法是标准霍夫变换（SHT）算法的一个改进，它在一定的范围内进行霍夫变换，计算单独线段的方向以及范围，从而减少计算量，缩短计算时间。之所以称 PPHT 为“概率”的，是因为并不将累加器平面内的所有可能的点累加，而只是累加其中的一部分，该想法是如果峰值如果足够高，只用一小部分时间去寻找它就够了。按照猜想，可以实质性地减少计算时间。

在 OpenCV 中，可以用 HoughLines 函数来调用标准霍夫变换（SHT）和多尺度霍夫变换（MSHT）。

而 HoughLinesP 函数用于调用累计概率霍夫变换 PPHT。累计概率霍夫变换执行效率很高，所有相比于 HoughLines 函数，我们更倾向于使用 HoughLinesP 函数。

总结一下，OpenCV 中的霍夫线变换有如下三种：

- 标准霍夫变换（StandardHough Transform, SHT），由 HoughLines 函数调用。
- 多尺度霍夫变换（Multi-ScaleHough Transform, MSHT），由 HoughLines 函数调用。
- 累计概率霍夫变换（ProgressiveProbabilistic Hough Transform, PPHT），由 HoughLinesP 函数调用。

## 7.2.3 霍夫线变换的原理

(1) 众所周知，一条直线在图像二维空间可由两个变量表示，有以下两种情况。如图 7.16 所示。

① 在笛卡尔坐标系：可由参数斜率和截距（ $m, b$ ）表示。

② 在极坐标系：可由参数极径和极角（ $r, \theta$ ）表示。

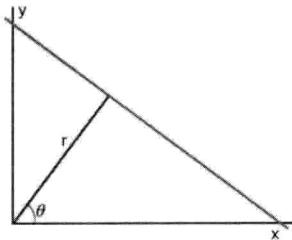


图 7.16 一条直线的两种表示方法

对于霍夫变换，我们将采用第二种方式极坐标系来表示直线。因此，直线的表达式可为：

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

化简便可得到：

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

(2) 一般来说对于点  $(x_0, y_0)$ ，可以将通过这个点的一族直线统一定义为：

$$r_0 = x_0 \cdot \cos \theta + y_0 \cdot \sin \theta$$

这就意味着每一对  $(r_0, \theta)$  代表一条通过点  $(x_0, y_0)$  的直线。

(3) 如果对于一个给定点  $(x_0, y_0)$ ，我们在极坐标对极径极角平面绘出所有通过它的直线，将得到一条正弦曲线。例如，对于给定点  $x_0=8$  和  $y_0=6$  可以绘出如 7.17 所示平面图。

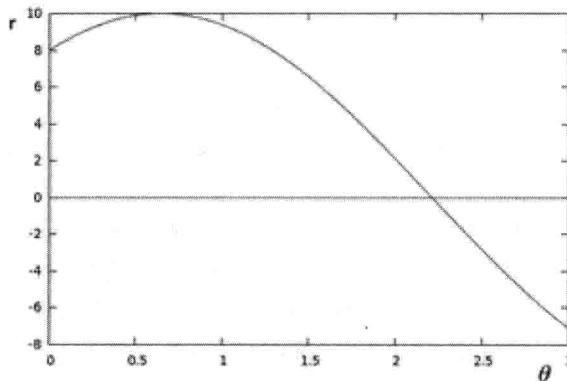


图 7.17 绘制出的正弦曲线

只绘出满足下列条件的点  $r > 0$  和  $0 < \theta < 2\pi$

(4) 我们可以对图像中所有的点进行上述操作。如果两个不同点进行上述操作后得到的曲线在平面  $\theta - r$  相交，这就意味着它们通过同一条直线。例如，接上面的例子继续对点  $x_1=9, y_1=4$  和点  $x_2=12, y_2=3$  绘图，得到图 7.18。

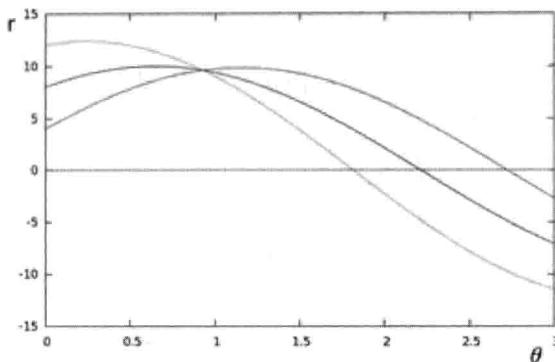


图 7.18 绘制出的曲线

这三条曲线在平面相交于点  $(0.925, 9.6)$ ，坐标表示的是参数对  $\theta - r$  或者是说点  $(x_0, y_0)$ ，点  $(x_1, y_1)$  和点  $(x_2, y_2)$  组成的平面内的直线。

(5) 以上的说明表明，一般来说，一条直线能够通过在平面  $\theta - r$  寻找交于一点的曲线数量来检测。而越多曲线交于一点也就意味着这个交点表示的直线由更多的点组成。一般来说我们可以通过设置直线上点的阈值来定义多少条曲线交于一点，这样才认为检测到了一条直线。

(6) 这就是霍夫线变换要做的。它追踪图像中每个点对应曲线间的交点。如果交于一点的曲线的数量超过了阈值，那么可以认为这个交点所代表的参数对  $(\theta, r_\theta)$  在原图像中为一条直线。

#### 7.2.4 标准霍夫变换：HoughLines()函数

此函数可以找出采用标准霍夫变换的二值图像线条。在 OpenCV 中，我们可以用其来调用标准霍夫变换 SHT 和多尺度霍夫变换 MSHT 的 OpenCV 内建算法。

```
C++: void HoughLines(InputArray image, OutputArray lines, double rho,
double theta, int threshold, double srn=0, double stn=0 )
```

- 第一个参数，`InputArray` 类型的 `image`，输入图像，即源图像。需为 8 位的单通道二进制图像，可以将任意的源图载入进来，并由函数修改成此格式后，再填在这里。
- 第二个参数，`InputArray` 类型的 `lines`，经过调用 `HoughLines` 函数后储存了霍夫线变换检测到线条的输出矢量。每一条线由具有两个元素的矢量  $(\rho, \theta)$  表示，其中， $\rho$  是离坐标原点  $(0,0)$ （也就是图像的左上角）的距离， $\theta$  是弧度线条旋转角度（0 度表示垂直线， $\pi/2$  度表示水平线）。
- 第三个参数，`double` 类型的 `rho`，以像素为单位的距离精度。另一种表述方式是直线搜索时的进步尺寸的单位半径。（Latex 中 `/rho` 即表示  $\rho$ ）
- 第四个参数，`double` 类型的 `theta`，以弧度为单位的角度精度。另一种表述方式是直线搜索时的进步尺寸的单位角度。
- 第五个参数，`int` 类型的 `threshold`，累加平面的阈值参数，即识别某部分为图中的一条直线时它在累加平面中必须达到的值。大于阈值 `threshold` 的线

段才可以被检测通过并返回到结果中。

- 第六个参数，double 类型的 srn，有默认值 0。对于多尺度的霍夫变换，这是第三个参数进步尺寸 rho 的除数距离。粗略的累加器进步尺寸直接是第三个参数 rho，而精确的累加器进步尺寸为 rho/srn。
- 第七个参数，double 类型的 stn，有默认值 0，对于多尺度霍夫变换，srn 表示第四个参数进步尺寸的单位角度 theta 的除数距离。且如果 srn 和 stn 同时为 0，就表示使用经典的霍夫变换。否则，这两个参数应该都为正数。

学完函数解析后，看一个以 HoughLines 为核心的示例程序，就可以全方位了解 HoughLines 函数的使用方法。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----  
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace cv;
using namespace std;  
  
-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----  
int main()
{
    //【1】载入原始图和 Mat 变量定义
    Mat srcImage = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材
    图
    Mat midImage,dstImage;//临时变量和目标图的定义  
  
    //【2】进行边缘检测和转化为灰度图
    Canny(srcImage, midImage, 50, 200, 3); //进行一次 canny 边缘检测
    cvtColor(midImage,dstImage, CV_GRAY2BGR); //转化边缘检测后的图为灰度图  
  
    //【3】进行霍夫线变换
    vector<Vec2f> lines; //定义一个矢量结构 lines 用于存放得到的线段矢量集合
    HoughLines(midImage, lines, 1, CV_PI/180, 150, 0, 0 );  
  
    //【4】依次在图中绘制出每条线段
    for( size_t i = 0; i < lines.size(); i++ )
    {
        float rho = lines[i][0], theta = lines[i][1];
        Point pt1, pt2;
        double a = cos(theta), b = sin(theta);
        double x0 = a*rho, y0 = b*rho;
        pt1.x = cvRound(x0 + 1000*(-b));
        pt1.y = cvRound(y0 + 1000*(a));
        pt2.x = cvRound(x0 - 1000*(-b));
        pt2.y = cvRound(y0 - 1000*(a));
        //此句代码的 OpenCV2 版为：
        //line( dstImage, pt1, pt2, Scalar(55,100,195), 1, CV_AA);
```

```

    //此句代码的 OpenCV3 版为：
    line( dstImage, pt1, pt2, Scalar(55,100,195), 1, LINE_AA);
}

//【5】显示原始图
imshow("【原始图】", srcImage);

//【6】边缘检测后的图
imshow("【边缘检测后的图】", midImage);

//【7】显示效果图
imshow("【效果图】", dstImage);

waitKey(0);

return 0;
}

```

运行截图如图 7.19 所示。

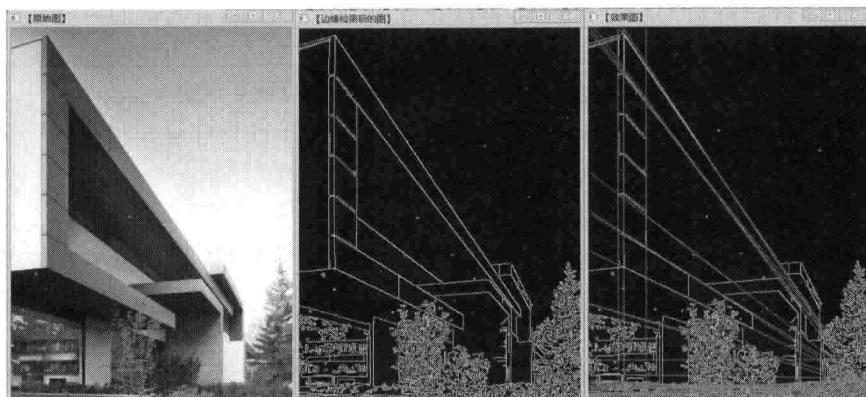


图 7.19 标准霍夫线变换效果图



可以通过调节 `line(dstImage, pt1, pt2, Scalar(55,100,195), 1, CV_AA);` 一句 `Scalar(55,100,195)` 参数中 G、B、R 颜色值的数值，得到图中想要的线条颜色。

### 7.2.5 累计概率霍夫变换：HoughLinesP()函数

此函数在 HoughLines 的基础上，在末尾加了一个代表 Probabilistic（概率）的 P，表明它可以采用累计概率霍夫变换（PPHT）来找出二值图像中的直线。

C++: void HoughLinesP(InputArray image, OutputArray lines, double rho,  
double theta, int threshold, double minLineLength=0, double  
maxLineGap=0 )

- 第一个参数，InputArray 类型的 image，输入图像，即源图像。需为 8 位的单通道二进制图像，可以将任意的源图载入进来后由函数修改成此格式后，再填在这里。

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

- 第二个参数，`InputArray` 类型的 `lines`，经过调用 `HoughLinesP` 函数后存储了检测到的线条的输出矢量，每一条线由具有 4 个元素的矢量( $x_1, y_1, x_2, y_2$ ) 表示，其中， $(x_1, y_1)$  和  $(x_2, y_2)$  是每个检测到的线段的结束点。
- 第三个参数，`double` 类型的 `rho`，以像素为单位的距离精度。另一种表达方式是直线搜索时的进步尺寸的单位半径。
- 第四个参数，`double` 类型的 `theta`，以弧度为单位的角度精度。另一种表达方式是直线搜索时的进步尺寸的单位角度。
- 第五个参数，`int` 类型的 `threshold`，累加平面的阈值参数，即识别某部分为图中的一条直线时它在累加平面中必须达到的值。大于阈值 `threshold` 的线段才可以被检测通过并返回到结果中。
- 第六个参数，`double` 类型的 `minLineLength`，有默认值 0，表示最低线段的长度，比这个设定参数短的线段就不能被显现出来。
- 第七个参数，`double` 类型的 `maxLineGap`，有默认值 0，允许将同一行点与点之间连接起来的最大的距离。

对于此函数，依然是为大家准备了示例程序，如下。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace cv;
using namespace std;

-----【main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
-----
int main()
{
    //【1】载入原始图和 Mat 变量定义
    Mat srcImage = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材
图
    Mat midImage,dstImage;//临时变量和目标图的定义

    //【2】进行边缘检测和转化为灰度图
    Canny(srcImage, midImage, 50, 200, 3);//进行一次 canny 边缘检测
    cvtColor(midImage,dstImage, COLOR_GRAY2BGR );//转化边缘检测后的图为灰
度图

    //【3】进行霍夫线变换
    vector<Vec4i> lines;//定义一个矢量结构 lines 用于存放得到的线段矢量集合
    HoughLinesP(midImage, lines, 1, CV_PI/180, 80, 50, 10 );

    //【4】依次在图中绘制出每条线段
    for( size_t i = 0; i < lines.size(); i++ )
    {
        Vec4i l = lines[i];
```

```

//此句代码的 OpenCV2 版为：
    //line( dstImage, Point(l[0], l[1]), Point(l[2], l[3]),
    Scalar(186,88,255), 1, CV_AA);
    //此句代码的 OpenCV3 版为：
    line( dstImage, Point(l[0], l[1]), Point(l[2], l[3]),
    Scalar(186,88,255), 1, LINE_AA);
}

//【5】显示原始图
imshow("【原始图】", srcImage);

//【6】边缘检测后的图
imshow("【边缘检测后的图】", midImage);

//【7】显示效果图
imshow("【效果图】", dstImage);

waitKey(0);

return 0;
}

```

运行截图如图 7.20 所示。

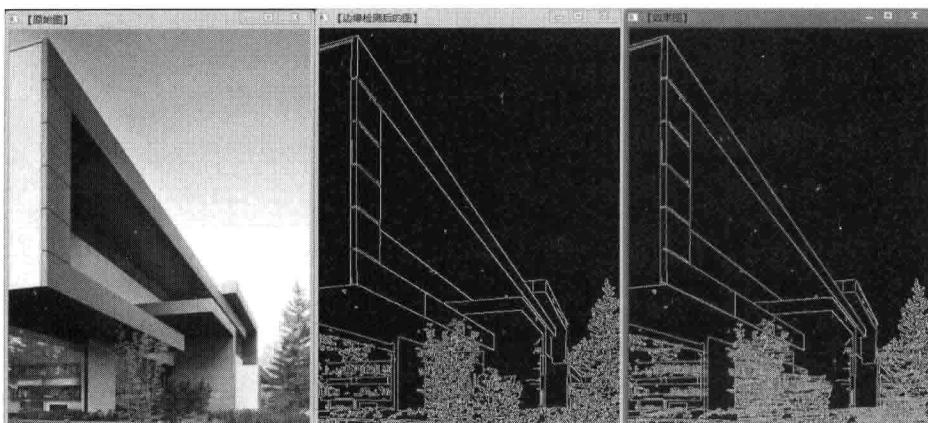


图 7.20 累计概率霍夫变换效果图

### 7.2.6 霍夫圆变换

霍夫圆变换的基本原理和上面讲的霍夫线变化大体上是很类似的，只是点对应的二维极径极角空间被三维的圆心点  $x$ 、 $y$  和半径  $r$  空间取代。说“大体上类似”的原因是，如果完全用相同的方法的话，累加平面会被三维的累加容器所代替——在这三维中，一维是  $x$ ，一维是  $y$ ，另外一维是圆的半径  $r$ 。这就意味着需要大量的内存而且执行效率会很低，速度会很慢。

对直线来说，一条直线能由参数极径极角  $(r, \theta)$  表示。而对圆来说，我们需要三个参数来表示一个圆，也就是：

$$C: (x_{center}, y_{center}, r)$$

这里的  $(x_{center}, y_{center})$  表示圆心的位置（下图中球心的点），而  $r$  表示半径。这样我们就能唯一的定义一个圆了，见下图。



7.21 原始图



7.22 霍夫圆变换效果图

在 OpenCV 中，我们常常通过一个叫做“霍夫梯度法”的方法来解决圆变换的问题。

### 7.2.7 霍夫梯度法的原理

霍夫梯度法的原理是这样的：

- (1) 首先对图像应用边缘检测，比如用 canny 边缘检测。
- (2) 然后，对边缘图像中的每一个非零点，考虑其局部梯度，即用 Sobel() 函数计算 x 和 y 方向的 Sobel 一阶导数得到梯度。
- (3) 利用得到的梯度，由斜率指定的直线上的每一个点都在累加器中被累加，这里的斜率是从一个指定的最小值到指定的最大值的距离。
- (4) 同时，标记边缘图像中每一个非 0 像素的位置。
- (5) 然后从二维累加器中这些点中选择候选的中心，这些中心都大于给定阈值并且大于其所有近邻。这些候选的中心按照累加值降序排列，以便于最支持像素的中心首先出现。
- (6) 接下来对每一个中心，考虑所有的非 0 像素。
- (7) 这些像素按照其与中心的距离排序。从到最大半径的最小距离算起，选择非 0 像素最支持的一条半径。
- (8) 如果一个中心收到边缘图像非 0 像素最充分的支持，并且到前期被选择的中心有足够的距离，那么它就会被保留下来。

这个实现可以使算法执行起来更高效，或许更加重要的是，能够帮助解决三维累加器中会产生许多噪声并且使得结果不稳定的稀疏分布问题。

人无完人，金无足赤。同样，这个算法也并不是十全十美的，还有许多需要指出的缺点。

### 7.2.8 霍夫梯度法的缺点

(1) 在霍夫梯度法中，我们使用 Sobel 导数来计算局部梯度，那么随之而来的假设是，它可以视作等同于一条局部切线，这并不是一个数值稳定的做法。在大多数情况下，这样做会得到正确的结果，但或许会在输出中产生一些噪声。

(2) 在边缘图像中的整个非 0 像素集被看做每个中心的候选部分。因此，如果把累加器的阈值设置偏低，算法将要消耗比较长的时间。此外，因为每一个中心只选择一个圆，如果有同心圆，就只能选择其中的一个。

(3) 因为中心是按照其关联的累加器值的升序排列的，并且如果新的中心过于接近之前已经接受的中心的话，就不会被保留下来。且当有许多同心圆或者是近似的同心圆时，霍夫梯度法的倾向是保留最大的一个圆。可以说这是一种比较极端的做法，因为在这里默认 Sobel 导数会产生噪声，若是对于无穷分辨率的平滑图像而言的话，这才是必须的。

### 7.2.9 霍夫圆变换：HoughCircles()函数

HoughCircles 函数可以利用霍夫变换算法检测出灰度图中的圆。它相比之前的 HoughLines 和 HoughLinesP，比较明显的一个区别是不需要源图是二值的，而 HoughLines 和 HoughLinesP 都需要源图为二值图像。

```
C++: void HoughCircles(InputArray image, OutputArray circles, int method,  
double dp, double minDist, double param1=100, double param2=100, int  
minRadius=0, int maxRadius=0 )
```

- 第一个参数，InputArray 类型的 image，输入图像，即源图像，需为 8 位的灰度单通道图像。
- 第二个参数，InputArray 类型的 circles，经过调用 HoughCircles 函数后此参数存储了检测到的圆的输出矢量，每个矢量由包含了 3 个元素的浮点矢量 (x,y,radius) 表示。
- 第三个参数，int 类型的 method，即使用的检测方法，目前 OpenCV 中就霍夫梯度法一种可以使用，它的标识符为 HOUGH\_GRADIENT (OpenCV2 中可写作 CV\_HOUGH\_GRADIENT)，在此参数处填这个标识符即可。
- 第四个参数，double 类型的 dp，用来检测圆心的累加器图像的分辨率于输入图像之比的倒数，且此参数允许创建一个比输入图像分辨率低的累加器。例如，如果  $dp=1$  时，累加器和输入图像具有相同的分辨率。如果  $dp=2$ ，累加器便有输入图像一半那么大的宽度和高度。
- 第五个参数，double 类型的 minDist，为霍夫变换检测到的圆的圆心之间的最小距离，即让算法能明显区分的两个不同圆之间的最小距离。这个参数如果太小的话，多个相邻的圆可能被错误地检测成了一个重合的圆。反之，这个参数设置太大，某些圆就不能被检测出来。
- 第六个参数，double 类型的 param1，有默认值 100。它是第三个参数 method 设置的检测方法的对应的参数。对当前唯一的方法霍夫梯度法

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

CV\_HOUGH\_GRADIENT，它表示传递给 canny 边缘检测算子的高阈值，而低阈值为高阈值的一半。

- 第七个参数，double 类型的 param2，也有默认值 100。它是第三个参数 method 设置的检测方法的对应的参数。对当前唯一的方法霍夫梯度法 CV\_HOUGH\_GRADIENT，它表示在检测阶段圆心的累加器阈值。它越小，就越可以检测到更多根本不存在的圆，而它越大的话，能通过检测的圆就更加接近完美的圆形了。
- 第八个参数，int 类型的 minRadius，有默认值 0，表示圆半径的最小值。
- 第九个参数，int 类型的 maxRadius，也有默认值 0，表示圆半径的最大值。

需要注意的是，使用此函数可以很容易地检测出圆的圆心，但是它可能找不到合适的圆半径。我们可以通过第八个参数 minRadius 和第九个参数 maxRadius 指定最小和最大的圆半径，来辅助圆检测的效果。或者，可以直接忽略返回半径，因为它们都有着默认值 0，只用 HoughCircles 函数检测出来的圆心，然后用额外的一些步骤来进一步确定半径。

依然是为大家准备了基于此函数的示例程序。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
-----
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace cv;
using namespace std;

-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
-----
int main()
{
    //【1】载入原始图、Mat 变量定义
    Mat srcImage = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材
    图
    Mat midImage, dstImage; //临时变量和目标图的定义

    //【2】显示原始图
    imshow("【原始图】", srcImage);

    //【3】转为灰度图并进行图像平滑
    cvtColor(srcImage, midImage, COLOR_BGR2GRAY); //转化边缘检测后的图为灰
    度图
    GaussianBlur( midImage, midImage, Size(9, 9), 2, 2 );

    //【4】进行霍夫圆变换
    vector<Vec3f> circles;
    HoughCircles( midImage, circles, HOUGH_GRADIENT, 1.5, 10, 200, 100,
    0, 0 );
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

//【5】依次在图中绘制出圆
for( size_t i = 0; i < circles.size(); i++ )
{
    //参数定义
    Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
    int radius = cvRound(circles[i][2]);
    //绘制圆心
    circle( srcImage, center, 3, Scalar(0,255,0), -1, 8, 0 );
    //绘制圆轮廓
    circle( srcImage, center, radius, Scalar(155,50,255), 3, 8, 0 );
}

//【6】显示效果图
imshow("【效果图】", srcImage);

waitKey(0);

return 0;
}

```

程序运行截图见图 7.21、7.22。

### 7.2.10 综合示例：霍夫变换

这次的综合示例，我们在 HoughLinesP 函数的基础上，为其添加了用于控制其第五个参数阈值 threshold 的滚动条，因此可以通过调节滚动条来改变阈值，从而动态地控制霍夫线变换检测的线条多少。

详细注释的程序源代码如下。

```

//-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace cv;

//-----【全局变量声明部分】-----
//      描述：全局变量声明
//-----

Mat g_srcImage, g_dstImage,g_midImage;//原始图、中间图和效果图
vector<Vec4i> g_lines;//定义一个矢量结构 g_lines 用于存放得到的线段矢量集合
//变量接收的 TrackBar 位置参数
int g_nthreshold=100;

//-----【全局函数声明部分】-----
//      描述：全局函数声明
//-----

```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
static void on_HoughLines(int, void*);//回调函数

//-----【main()函数】-----
//    描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----
int main()
{
    //改变 console 字体颜色
    system("color 3F");

    //载入原始图和 Mat 变量定义
    Mat g_srcImage = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的
    素材图

    //显示原始图
    imshow("【原始图】", g_srcImage);

    //创建滚动条
    namedWindow("【效果图】",1);
    createTrackbar("值", "【效果图】", &g_nthreshold, 200, on_HoughLines);

    //进行边缘检测和转化为灰度图
    Canny(g_srcImage, g_midImage, 50, 200, 3); //进行一次 canny 边缘检测
    cvtColor(g_midImage,g_dstImage, COLOR_GRAY2BGR); //转化边缘检测后的图
    为灰度图

    //调用一次回调函数，调用一次 HoughLinesP 函数
    on_HoughLines(g_nthreshold,0);
    HoughLinesP(g_midImage, g_lines, 1, CV_PI/180, 80, 50, 10 );

    //显示效果图
    imshow("【效果图】", g_dstImage);

    waitKey(0);

    return 0;
}

//-----【on_HoughLines()函数】-----
//    描述：【顶帽运算/黑帽运算】窗口的回调函数
//-----
static void on_HoughLines(int, void*)
{
    //定义局部变量储存全局变量
    Mat dstImage=g_dstImage.clone();
    Mat midImage=g_midImage.clone();

    //调用 HoughLinesP 函数
    vector<Vec4i> mylines;
    HoughLinesP(midImage, mylines, 1, CV_PI/180, g_nthreshold+1, 50,
```

```
10 );  
  
    //循环遍历绘制每一条线段  
    for( size_t i = 0; i < mylines.size(); i++ )  
    {  
        Vec4i l = mylines[i];  
        line( dstImage, Point(l[0], l[1]), Point(l[2], l[3]),  
              Scalar(23,180,55), 1, LINE_AA);  
    }  
    //显示图像  
    imshow("【效果图】",dstImage);  
}
```

放一些运行截图。如图 7.23~7.26 所示。

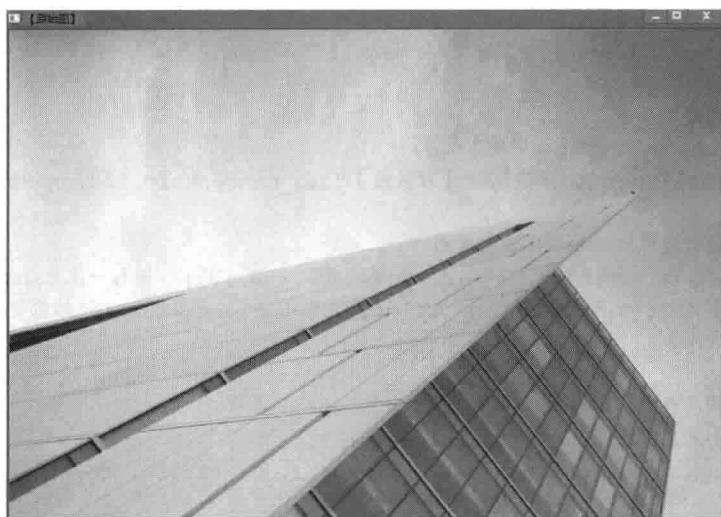


图 7.23 原始图



图 7.24 阈值为 95 时的程序截图

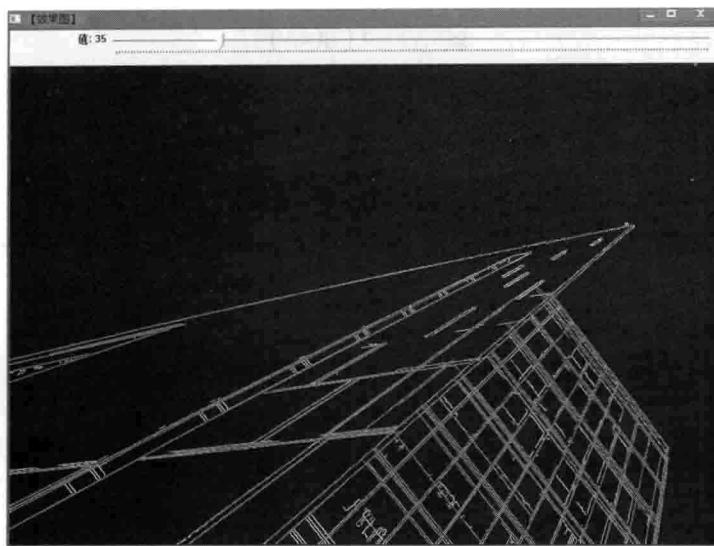


图 7.25 阈值为 35 时的程序截图

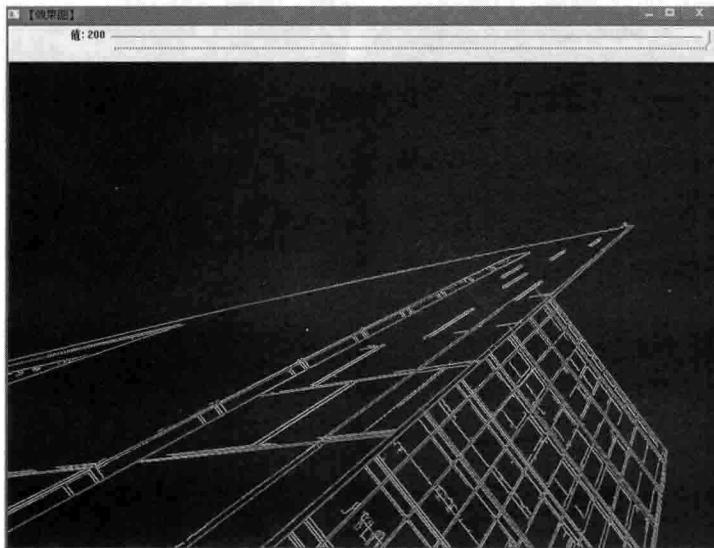


图 7.26 阈值为 200 时的程序截图

## 7.3 重映射

本节中，我们主要一起了解重映射的概念，以及 OpenCV 中相关的实现函数 `remap()`。

### 7.3.1 重映射的概念

重映射，就是把一幅图像中某位置的像素放置到另一个图片指定位置的过程。为了完成映射过程，需要获得一些插值为非整数像素的坐标，因为源图像与目标图像的像素坐标不是一一对应的。一般情况下，我们通过重映射来表达每个像素的位置  $(x, y)$ ，像这样：

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

在这里,  $g()$ 是目标图像,  $f()$ 是源图像, 而  $h(x,y)$ 是作用于 $(x,y)$ 的映射方法函数。

来看个例子。若有一幅图像 I, 对其按照下面的条件作重映射:

$$h(x,y) = (I.cols - x, y)$$

图像会按照 x 轴方向发生翻转。那么, 源图像和效果图分别如图 7.27 和 7.28 所示。



图 7.27 原始图



图 7.28 经过重映射的效果图

在 OpenCV 中, 可以使用函数 `remap()`来实现简单重映射, 下面我们就一起来看看这个函数。

### 7.3.2 实现重映射: `remap()`函数

`remap()`函数会根据指定的映射形式, 将源图像进行重映射几何变换, 基于的公式如下:

$$dst(x,y)=src(map_x(x,y),map_y(x,y))$$

需要注意, 此函数不支持就地 (in-place) 操作。看看其原型和参数。

```
C++: void remap(InputArray src, OutputArray dst, InputArray map1,
InputArray map2, int interpolation, int borderMode=BORDER_CONSTANT,
const Scalar& borderColor=Scalar())
```

- 第一个参数, `InputArray`类型的 `src`, 输入图像, 即源图像, 填 `Mat`类的对象即可, 且需为单通道 8 位或者浮点型图像。
- 第二个参数, `OutputArray`类型的 `dst`, 函数调用后的运算结果存在这里, 即这个参数用于存放函数调用后的输出结果, 需和源图片有一样的尺寸和类型。
- 第三个参数, `InputArray`类型的 `map1`, 它有两种可能的表示对象。
  - 表示点  $(x, y)$  的第一个映射。
  - 表示 `CV_16SC2`、`CV_32FC1` 或 `CV_32FC2` 类型的 X 值。
- 第四个参数, `InputArray`类型的 `map2`, 同样, 它也有两种可能的表示对象, 而且它会根据 `map1` 来确定表示那种对象。
  - 若 `map1` 表示点  $(x, y)$  时。这个参数不代表任何值。

- 表示 CV\_16UC1, CV\_32FC1 类型的 Y 值 (第二个值)。
- 第五个参数, int 类型的 interpolation, 插值方式, 之前的 resize() 函数中有讲到, 需要注意, resize() 函数中提到的 INTER\_AREA 插值方式在这里是不支持的, 所以可选的插值方式如下 (需要注意, 这些宏相应的 OpenCV2 版为在它们的宏名称前面加上 “CV\_” 前缀, 比如 “INTER\_LINEAR”的 OpenCV2 版为 “CV\_INTER\_LINEAR”):
  - INTER\_NEAREST——最近邻插值
  - INTER\_LINEAR——双线性插值 (默认值)
  - INTER\_CUBIC——双三次样条插值 (逾  $4 \times 4$  像素邻域内的双三次插值)
  - INTER\_LANCZOS4——Lanczos 插值 (逾  $8 \times 8$  像素邻域的 Lanczos 插值)
- 第六个参数, int 类型的 borderMode, 边界模式, 有默认值 BORDER\_CONSTANT, 表示目标图像中 “离群点 (outliers)” 的像素值不会被此函数修改。
- 第七个参数, const Scalar&类型的 borderValue, 当有常数边界时使用的值, 其有默认值 Scalar(), 即默认值为 0。

### 7.3.3 基础示例程序：基本重映射

下面将贴出精简后的以 remap 函数为核心的示例程序, 方便大家快速掌握 remap 函数的使用方法。

```
-----【头文件、命名空间包含部分】-----
//      描述: 包含程序所依赖的头文件和命名空间
-----

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;

-----【main() 函数】-----
//      描述: 控制台应用程序的入口函数, 我们的程序从这里开始执行
-----

int main( )
{
    //【0】变量定义
    Mat srcImage, dstImage;
    Mat map_x, map_y;

    //【1】载入原始图
    srcImage = imread( "1.jpg", 1 );
    if(!srcImage.data) { printf("读取图片错误, 请确定目录下是否有 imread 函
数指定的图片存在~! \n"); return false; }
    imshow("原始图", srcImage);

    //【2】创建和原始图一样的效果图, x 重映射图, y 重映射图
    dstImage.create( srcImage.size(), srcImage.type() );
    map_x.create( srcImage.size(), CV_32FC1 );
```

```
map_y.create( srcImage.size(), CV_32FC1 );

//【3】双层循环，遍历每一个像素点，改变 map_x & map_y 的值
for( int j = 0; j < srcImage.rows;j++)
{
    for( int i = 0; i < srcImage.cols;i++)
    {
        //改变 map_x & map_y 的值.
        map_x.at<float>(j,i) = static_cast<float>(i);
        map_y.at<float>(j,i) = static_cast<float>(srcImage.rows -
j);
    }
}

//【4】进行重映射操作
//此句代码的 OpenCV2 版为：
//remap( srcImage, dstImage, map_x, map_y, CV_INTER_LINEAR,
BORDER_CONSTANT, Scalar(0,0, 0) );
//此句代码的 OpenCV3 版为：
remap( srcImage, dstImage, map_x, map_y, INTER_LINEAR,
BORDER_CONSTANT, Scalar(0,0, 0) );
//【5】显示效果图
imshow( "【程序窗口】", dstImage );
waitKey();

return 0;
}
```

程序运行截图如图 7.29 和图 7.30 所示。



图 7.29 原始图

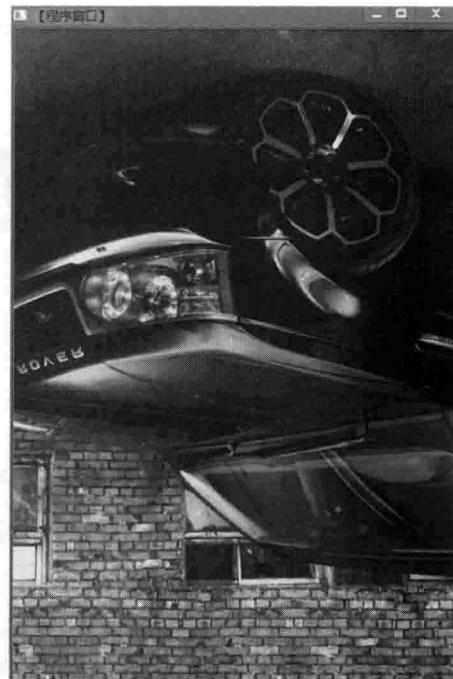


图 7.30 效果图

### 7.3.4 综合示例程序：实现多种重映射

先放出以 remap 为核心的综合示例程序，可以用按键控制四种不同的映射模式。如图 7.31 所示。

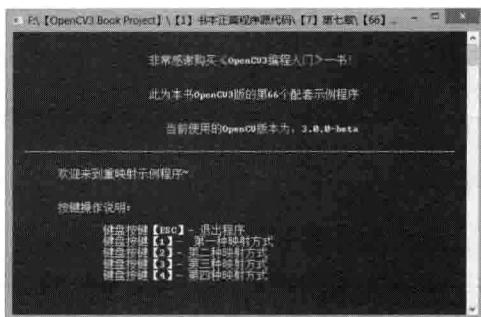


图 7.31 程序按键说明

这个程序详细注释的源代码如下。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所依赖的头文件和命名空间  
//-----  
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"  
#include <iostream>  
using namespace cv;  
using namespace std;  
  
-----【宏定义部分】-----  
//      描述：定义一些辅助宏  
//-----  
#define WINDOW_NAME "【程序窗口】"          //为窗口标题定义的宏  
  
-----【全局变量声明部分】-----  
//      描述：全局变量的声明  
//-----  
Mat g_srcImage, g_dstImage;  
Mat g_map_x, g_map_y;  
  
-----【全局函数声明部分】-----  
//      描述：全局函数的声明  
//-----  
int update_map( int key );  
static void ShowHelpText(); //输出帮助文字  
  
-----【main()函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行  
//-----  
int main( int argc, char** argv )  
{
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//改变 console 字体颜色
system("color 2F");

//显示帮助文字
ShowHelpText();

//【1】载入原始图
g_srcImage = imread( "1.jpg", 1 );
if(!g_srcImage.data ) { printf("读取图片错误, 请确定目录下是否有 imread\n函数指定的图片存在~! \n"); return false; }
imshow("原始图", g_srcImage);

//【2】创建和原始图一样的效果图, x 重映射图, y 重映射图
g_dstImage.create( g_srcImage.size(), g_srcImage.type() );
g_map_x.create( g_srcImage.size(), CV_32FC1 );
g_map_y.create( g_srcImage.size(), CV_32FC1 );

//【3】创建窗口并显示
namedWindow( WINDOW_NAME, WINDOW_AUTOSIZE );
imshow(WINDOW_NAME, g_srcImage);

//【4】轮询按键, 更新 map_x 和 map_y 的值, 进行重映射操作并显示效果图
while( 1 )
{
    //获取键盘按键
    int key = waitKey(0);

    //判断 ESC 是否按下, 若按下便退出
    if( (key & 255) == 27 )
    {
        cout << "程序退出.....\n";
        break;
    }

    //根据按下的键盘按键来更新 map_x & map_y 的值. 然后调用 remap()进行重映射
    update_map(key);
    //此句代码的 OpenCV 版为:
    //remap( g_srcImage, g_dstImage, g_map_x, g_map_y,
    //CV_INTER_LINEAR, BORDER_CONSTANT, Scalar(0,0, 0) );
    //此句代码的 OpenCV3 版为:
    remap( g_srcImage, g_dstImage, g_map_x, g_map_y, INTER_LINEAR,
    BORDER_CONSTANT, Scalar(0,0, 0) );
    //显示效果图
    imshow( WINDOW_NAME, g_dstImage );
}

return 0;
}

-----【 update_map() 函数 】-----
//      描述: 根据按键来更新 map_x 与 map_y 的值
-----
int update_map( int key )
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
{  
    //双层循环，遍历每一个像素点  
    for( int j = 0; j < g_srcImage.rows;j++)  
    {  
        for( int i = 0; i < g_srcImage.cols;i++)  
        {  
            switch(key)  
            {  
                case '1': // 键盘【1】键按下，进行第一种重映射操作  
                    if( i > g_srcImage.cols*0.25 && i < g_srcImage.cols*0.75  
&& j > g_srcImage.rows*0.25 && j < g_srcImage.rows*0.75)  
                    {  
                        g_map_x.at<float>(j,i) = static_cast<float>(2*( i -  
g_srcImage.cols*0.25 ) + 0.5);  
                        g_map_y.at<float>(j,i) = static_cast<float>(2*( j -  
g_srcImage.rows*0.25 ) + 0.5);  
                    }  
                    else  
                    {  
                        g_map_x.at<float>(j,i) = 0;  
                        g_map_y.at<float>(j,i) = 0;  
                    }  
                    break;  
                case '2':// 键盘【2】键按下，进行第二种重映射操作  
                    g_map_x.at<float>(j,i) = static_cast<float>(i);  
                    g_map_y.at<float>(j,i) =  
static_cast<float>(g_srcImage.rows - j);  
                    break;  
                case '3':// 键盘【3】键按下，进行第三种重映射操作  
                    g_map_x.at<float>(j,i) =  
static_cast<float>(g_srcImage.cols - i);  
                    g_map_y.at<float>(j,i) = static_cast<float>(j);  
                    break;  
                case '4':// 键盘【4】键按下，进行第四种重映射操作  
                    g_map_x.at<float>(j,i) =  
static_cast<float>(g_srcImage.cols - i);  
                    g_map_y.at<float>(j,i) =  
static_cast<float>(g_srcImage.rows - j);  
                    break;  
            }  
        }  
    }  
    return 1;  
}  
  
//-----【ShowHelpText() 函数】-----  
//      描述：输出一些帮助信息  
//-----  
static void ShowHelpText()  
{  
    //输出一些帮助信息  
    printf("\n\n\n\t欢迎来到重映射示例程序~\n\n");  
}
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
printf("\t 当前使用的 OpenCV 版本为 OpenCV "CV_VERSION);\nprintf( "\n\n\t 按键操作说明: \n\n\"\t\t 键盘按键【ESC】- 退出程序\n\"\t\t 键盘按键【1】- 第一种映射方式\n\"\t\t 键盘按键【2】- 第二种映射方式\n\"\t\t 键盘按键【3】- 第三种映射方式\n\"\t\t 键盘按键【4】- 第四种映射方式\n\" );\n}
```

程序的运行效果图如图 7.32~7.36 所示。



图 7.32 原始图



图 7.33 第一种重映射



图 7.34 第二种重映射

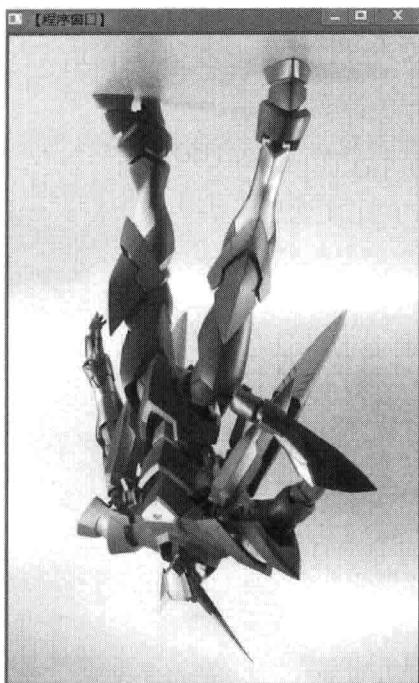


图 7.35 第三种重映射

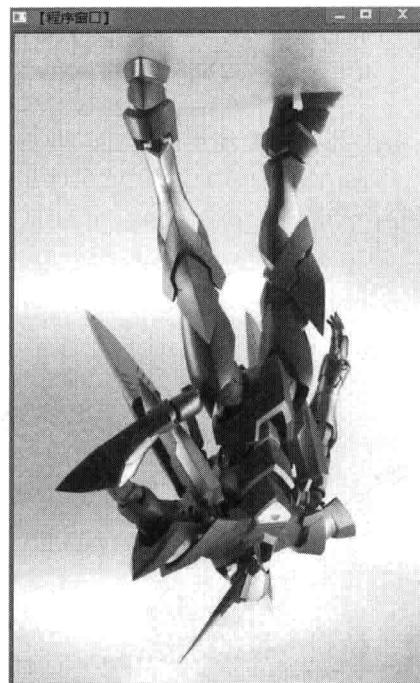


图 7.36 第四种重映射

## 7.4 仿射变换

本节中，我们将一起了解仿射变换的概念，以及 OpenCV 中相关的实现函数 `warpAffine` 和 `getRotationMatrix2D`。

### 7.4.1 认识仿射变换

仿射变换（Affine Transformation 或 Affine Map），又称仿射映射，是指在几何中，一个向量空间进行一次线性变换并接上一个平移，变换为另一个向量空间的过程。它保持了二维图形的“平直性”（直线经过变换之后依然是直线）和“平行性”（二维图形之间的相对位置关系保持不变，平行线依然是平行线，且直线上点的位置顺序不变）。

一个任意的仿射变换都能表示为乘以一个矩阵（线性变换）接着再加上一个向量（平移）的形式。

那么，我们能够用仿射变换来表示如下三种常见的变换形式：

- 旋转， rotation (线性变换)
- 平移， translation(向量加)
- 缩放， scale(线性变换)

进行更深层次的理解，仿射变换代表的是两幅图之间的一种映射关系。

而我们通常使用  $2 \times 3$  的矩阵来表示仿射变换。

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}_{2 \times 2} \quad B = \begin{bmatrix} b_{00} \\ b_{10} \end{bmatrix}_{2 \times 1} \quad M = [A \quad B] = \begin{bmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{10} \end{bmatrix}_{2 \times 3}$$

考虑到我们要使用矩阵  $A$  和  $B$  对二维向量  $X = \begin{bmatrix} x \\ y \end{bmatrix}$  做变换，所以也能表示为下列形式。

$$T = A \cdot \begin{bmatrix} x \\ y \end{bmatrix} + B$$

或者

$$T = M \cdot [x, y, 1]^T$$

即：

$$T = \begin{bmatrix} a_{00}x + a_{01}y + b_{00} \\ a_{10}x + a_{11}y + b_{10} \end{bmatrix}_{2 \times 2}$$

### 7.4.2 仿射变换的求法

我们知道，仿射变换表示的就是两幅图片之间的一种联系，关于这种联系的信息大致可从以下两种场景获得。

- 已知  $X$  和  $T$ ，而且已知它们是有联系的。接下来的工作就是求出矩阵  $M$ 。
- 已知  $M$  和  $X$ ，想求得  $T$ 。只要应用算式  $T = M \cdot X$  即可。对于这种联系的信息可以用矩阵  $M$  清晰地表达（即给出明确的  $2 \times 3$  矩阵），也可以用两幅图片点之间几何关系来表达。

形象地说明一下，因为矩阵  $M$  联系着两幅图片，就以其表示两图中各三点直接的联系为例。如图 7.37 所示。

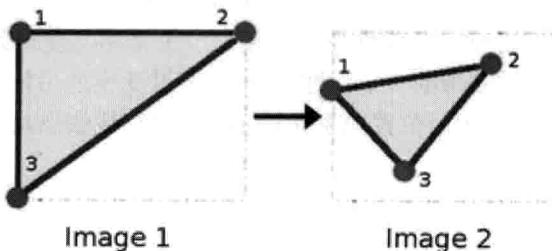


图 7.37 示意图

图中，点 1、2 和 3（在 Image1 中形成一个三角形）与 Image2 中的三个点是一一映射的关系，且它们仍然形成三角形，但形状已经和之前不一样了。我们能通过这样两组三点求出仿射变换（可以选择自己喜欢的点），接着就可以把仿射变换应用到图像中去。

OpenCV 仿射变换相关的函数一般涉及到 warpAffine 和 getRotationMatrix2D 这两个函数：

- 使用 OpenCV 函数 warpAffine 来实现一些简单的重映射。
- 使用 OpenCV 函数 getRotationMatrix2D 来获得旋转矩阵。

下面分别对其进行讲解。

### 7.4.3 进行仿射变换： warpAffine()函数

warpAffine 函数的作用是依据以下公式子，对图像做仿射变换。

$$dst(x, y) = src(M_{11}x + M_{12}y + M_{13}, M_{21}x + M_{22}y + M_{23})$$

函数原型如下。

```
C++: void warpAffine(InputArray src, OutputArray dst, InputArray M, Size dsize, int flags=INTER_LINEAR, int borderMode=BORDER_CONSTANT, const Scalar& borderColor=Scalar())
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可。
- 第二个参数，OutputArray 类型的 dst，函数调用后的运算结果存在这里，需和源图片有一样的尺寸和类型。
- 第三个参数，InputArray 类型的 M， $2 \times 3$  的变换矩阵。
- 第四个参数，Size 类型的 dsize，表示输出图像的尺寸。
- 第五个参数，int 类型的 flags，插值方法的标识符。此参数有默认值 INTER\_LINEAR(线性插值)，可选的插值方式如表 7.1 所示。

表 7.1 warpAffine 函数可选的插值方式

标识符	含义
INTER_NEAREST	最近邻插值
INTER_LINEAR	线性插值（默认值）
INTER_AREA	区域插值
INTER_CUBIC	三次样条插值
INTER_LANCZOS4	Lanczos 插值
CV_WARP_FILL_OUTLIERS	填充所有输出图像的象素。如果部分象素落在输入图像的边界外，那么它们的值设定为 fillval
CV_WARP_INVERSE_MAP	表示 M 为输出图像到输入图像的反变换。因此可以直接用来做象素插值。否则，warpAffine 函数从 M 矩阵得到反变换

- 第六个参数，int 类型的 borderMode，边界像素模式，默认值为 BORDER\_CONSTANT。
- 第七个参数，const Scalar&类型的 borderColor，在恒定的边界情况下取的值，默认值为 Scalar()，即 0。

另外提一点，WarpAffine 函数与一个叫做 cvGetQuadrangleSubPix()的函数类

似，但是不完全相同。WarpAffine 要求输入和输出图像具有同样的数据类型，有更大的资源开销（因此对小图像不太合适）而且输出图像的部分可以保留不变。而 cvGetQuadrangleSubPix 可以精确地从 8 位图像中提取四边形到浮点数缓存区中，具有比较小的系统开销，而且总是全部改变输出图像的内容。

#### 7.4.4 计算二维旋转变换矩阵：getRotationMatrix2D()函数

getRotationMatrix2D()函数用于计算二维旋转变换矩阵。变换会将旋转中心映射到它自身。

```
C++: Mat getRotationMatrix2D(Point2f center, double angle, double scale)
```

- 第一个参数，Point2f 类型的 center，表示源图像的旋转中心。
- 第二个参数，double 类型的 angle，旋转角度。角度为正值表示向逆时针旋转（坐标原点是左上角）。
- 第三个参数，double 类型的 scale，缩放系数。

此函数计算以下矩阵：

$$\begin{bmatrix} \alpha\beta(1-\alpha) \cdot \text{center}.x - \beta\text{center}.y \\ -\beta\alpha\beta \cdot \text{center}.x + (1-\alpha) \cdot \text{center}.y \end{bmatrix}$$

其中：

$$\begin{aligned} \alpha &= \text{scale} \cdot \cos \text{angle}, \\ \beta &= \text{scale} \cdot \sin \text{angle} \end{aligned}$$

#### 7.4.5 示例程序：仿射变换

学习完上面的讲解和函数实现，下面是一个以 warpAffine 和 getRotationMatrix2D 函数为核心的对图像进行仿射变换的示例程序。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----  

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;

-----【宏定义部分】-----
//      描述：定义一些辅助宏
//-----  

#define WINDOW_NAME1 "【原始图窗口】"           //为窗口标题定义的宏
#define WINDOW_NAME2 "【经过 Warp 后的图像】"     //为窗口标题定义的宏
#define WINDOW_NAME3 "【经过 Warp 和 Rotate 后的图像】" //为窗口标题定义的宏
```

```
-----【全局函数声明部分】-----
//      描述：全局函数的声明
-----
static void ShowHelpText();

-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
-----
int main( )
{
    //【0】改变 console 字体颜色
    system("color 1A");

    //【0】显示欢迎和帮助文字
    ShowHelpText();

    //【1】参数准备
    //定义两组点，代表两个三角形
    Point2f srcTriangle[3];
    Point2f dstTriangle[3];
    //定义一些 Mat 变量
    Mat rotMat( 2, 3, CV_32FC1 );
    Mat warpMat( 2, 3, CV_32FC1 );
    Mat srcImage, dstImage_warp, dstImage_warp_rotate;

    //【2】加载源图像并作一些初始化
    srcImage = imread( "l.jpg", 1 );
    if(!srcImage.data) { printf("读取图片错误，请确定目录下是否有 imread 函数指定的图片存在~! \n"); return false; }
    // 设置目标图像的大小和类型与源图像一致
    dstImage_warp = Mat::zeros( srcImage.rows, srcImage.cols,
    srcImage.type() );

    //【3】设置源图像和目标图像上的三组点以计算仿射变换
    srcTriangle[0] = Point2f( 0,0 );
    srcTriangle[1] = Point2f( static_cast<float>(srcImage.cols - 1),
    0 );
    srcTriangle[2] = Point2f( 0, static_cast<float>(srcImage.rows -
    1 ) );

    dstTriangle[0] = Point2f( static_cast<float>(srcImage.cols*0.0),
    static_cast<float>(srcImage.rows*0.33));
    dstTriangle[1] = Point2f( static_cast<float>(srcImage.cols*0.65),
    static_cast<float>(srcImage.rows*0.35));
    dstTriangle[2] = Point2f( static_cast<float>(srcImage.cols*0.15),
    static_cast<float>(srcImage.rows*0.6));

    //【4】求得仿射变换
    warpMat = getAffineTransform( srcTriangle, dstTriangle );

    //【5】对源图像应用刚刚求得的仿射变换
```

```
warpAffine( srcImage, dstImage_warp, warpMat,
dstImage_warp.size() );

//【6】对图像进行缩放后再旋转
// 计算绕图像中点顺时针旋转 50 度缩放因子为 0.6 的旋转矩阵
Point center = Point( dstImage_warp.cols/2, dstImage_warp.rows/2 );
double angle = -30.0;
double scale = 0.8;
// 通过上面的旋转细节信息求得旋转矩阵
rotMat = getRotationMatrix2D( center, angle, scale );
// 旋转已缩放后的图像

warpAffine( dstImage_warp, dstImage_warp_rotate, rotMat,
dstImage_warp.size() );

//【7】显示结果
imshow( WINDOW_NAME1, srcImage );
imshow( WINDOW_NAME2, dstImage_warp );
imshow( WINDOW_NAME3, dstImage_warp_rotate );

// 等待用户按任意按键退出程序
waitKey(0);

return 0;
}

//-----【ShowHelpText() 函数】-----
//      描述：输出一些帮助信息
//-----
static void ShowHelpText()
{
    //输出一些帮助信息
    printf( "\n\n\n\t欢迎来到【仿射变换】示例程序~\n\n");
    printf("\t当前使用的 OpenCV 版本为 OpenCV " CV_VERSION ); );
}
```

程序的运行截图如图 7.38、7.39、7.40 所示。



图 7.38 原始图

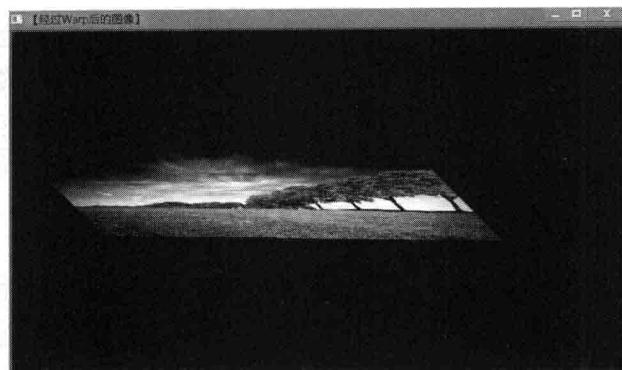


图 7.39 经过 Warp 后的图像

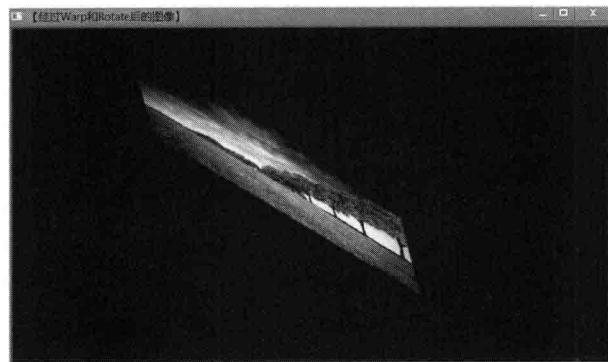


图 7.40 经过 Warp 和 Rotate 后的图像

## 7.5 直方图均衡化

很多时候，我们用相机拍摄的照片的效果往往会不尽人意。这时，可以对这些图像进行一些处理，来扩大图像的动态范围。这种情况下最常用到的技术就是直方图均衡化。未经均衡化的图片范例如图 7.41、7.42 所示。



图 7.41 未经均衡化的图像

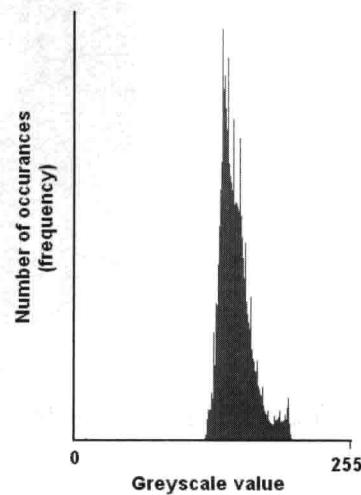


图 7.42 未经均衡化的图像相应的直方图

在图 7.41 中，我们可以看到，左边的图像比较淡，因为其数值范围变化比较小，可以在这幅图的直方图（图 7.42）中明显地看到。因为处理的是 8 位图像，其亮度值是从 0 到 255，但直方图值显示的实际亮度却集中在亮度范围的中间区域。为了解决这个问题，就可以用到直方图均衡化技术，先来看看其概念。

### 7.5.1 直方图均衡化的概念和特点

直方图均衡化是灰度变换的一个重要应用，它高效且易于实现，广泛应用于图像增强处理中。图像的像素灰度变化是随机的，直方图的图形高低不齐，直方图均衡化就是用一定的算法使直方图大致平和的方法。均衡化效果示例如图 7.43、7.44 所示。

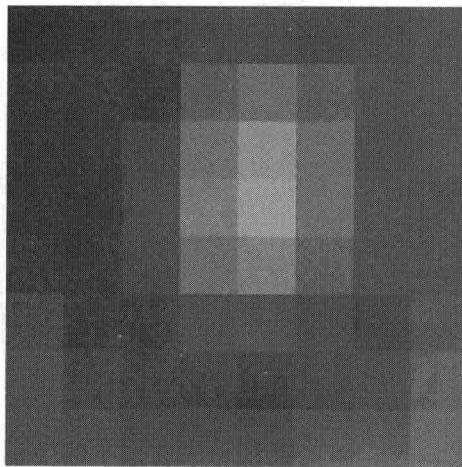


图 7.43 原始图

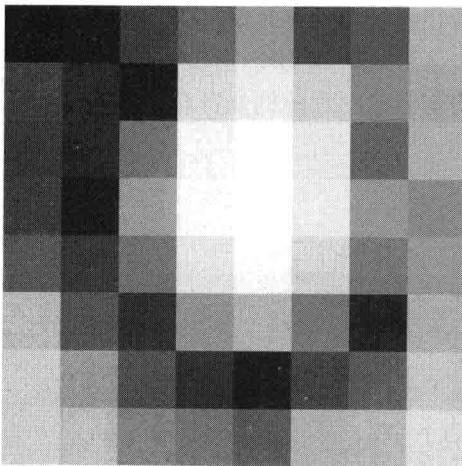


图 7.44 均衡化后图像

简而言之，直方图均衡化是通过拉伸像素强度分布范围来增强图像对比度的一种方法。

均衡化处理后的图像只能是近似均匀分布。均衡化图像的动态范围扩大了，

但其本质是扩大了量化间隔，而量化级别反而减少了，因此，原来灰度不同的像素经处理后可能变的相同，形成了一片相同灰度的区域，各区域之间有明显的边界，从而出现了伪轮廓。

在原始图像对比度本来就很高的情况下，如果再均衡化则灰度调和，对比度会降低。在泛白缓和的图像中，均衡化会合并一些像素灰度，从而增大对比度。均衡化后的图片如果再对其均衡化，则图像不会有任何变化。如图 7.45、7.46 所示。

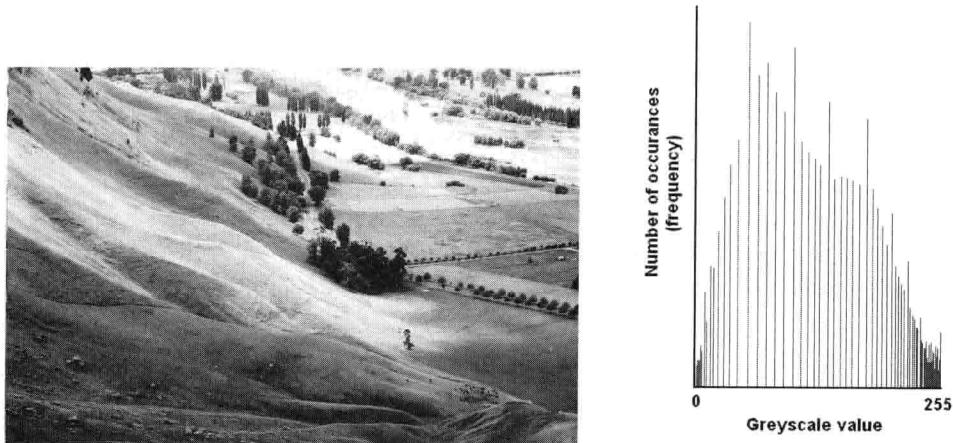


图 7.45 经过均衡化的同一幅图像

图 7.46 经过均衡化的图像相应的直方图

通过图 7.46 可以发现，经过均衡化的图像，其频谱更加舒展，有效地利用了 0~255 的空间，图像表现力更加出色。

### 7.5.2 实现直方图均衡化：equalizeHist()函数

在 OpenCV 中，直方图均衡化的功能实现由 equalizeHist 函数完成。我们一起看看它的函数描述。

```
C++: void equalizeHist(InputArray src, OutputArray dst)
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可，需为 8 位单通道的图像。
- 第二个参数，OutputArray 类型的 dst，函数调用后的运算结果存在这里，需和源图片有一样的尺寸和类型。

采用如下步骤对输入图像进行直方图均衡化。

- 1) 计算输入图像的直方图 H。
- 2) 进行直方图归一化，直方图的组距的和为 255。
- 3) 计算直方图积分：

$$H'(i) = \sum_{0 \leq j \leq i} H(j)$$

4) 以  $H'$  作为查询表进行图像变换:

$$dst(x,y)=H'(src(x,y))$$

言而言之, 由 `equalizeHist()` 函数实现的灰度直方图均衡化算法, 就是把直方图的每个灰度级进行归一化处理, 求每种灰度的累积分布, 得到一个映射的灰度映射表, 然后根据相应的灰度值来修正原图中的每个像素。

### 7.5.3 示例程序: 直方图均衡化

这一节将给大家演示的示例程序简明扼要而“字字珠玑”, 非常直观地演示出了如何用 `equalizeHist()` 函数来进行图像的直方图均衡化, 详细注释的代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述: 包含程序所使用的头文件和命名空间
//-----[ main() 函数 ]-----
//      描述: 控制台应用程序的入口函数, 我们的程序从这里开始执行
//-----
int main()
{
    // 【1】加载源图像
    Mat srcImage, dstImage;
    srcImage = imread( "l.jpg", 1 );
    if(!srcImage.data) { printf("读取图片错误, 请确定目录下是否有 imread 函数指定图片存在~! \n"); return false; }

    // 【2】转为灰度图并显示出来
    cvtColor( srcImage, srcImage, COLOR_BGR2GRAY );
    imshow( "原始图", srcImage );

    // 【3】进行直方图均衡化
    equalizeHist( srcImage, dstImage );

    // 【4】显示结果
    imshow( "经过直方图均衡化的图", dstImage );

    // 等待用户按键退出程序
    waitKey(0);
    return 0;
}
```

看完详细注释的代码, 一起看看程序运行后得到的窗口。如图 7.47、7.48 所示。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

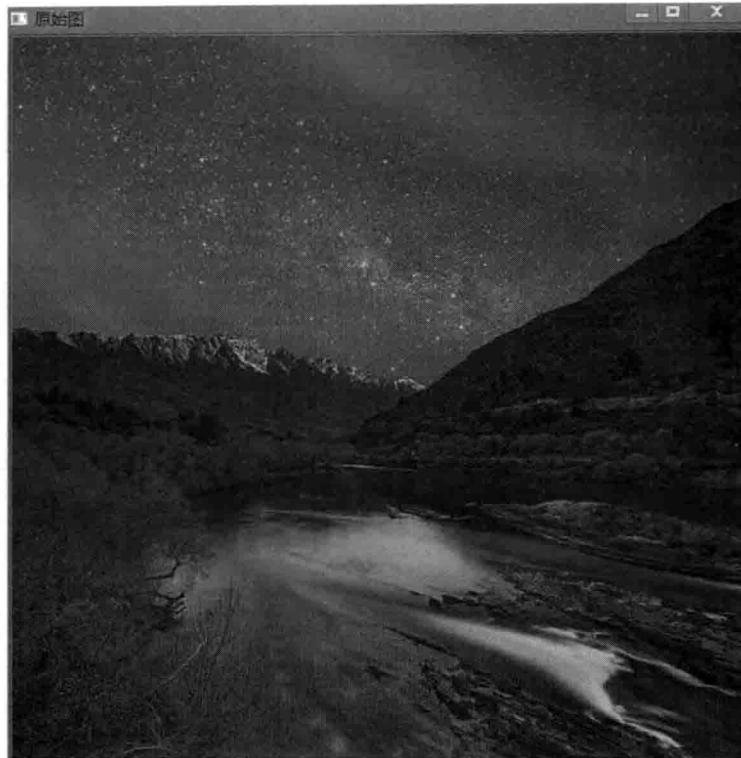


图 7.47 灰度素材图

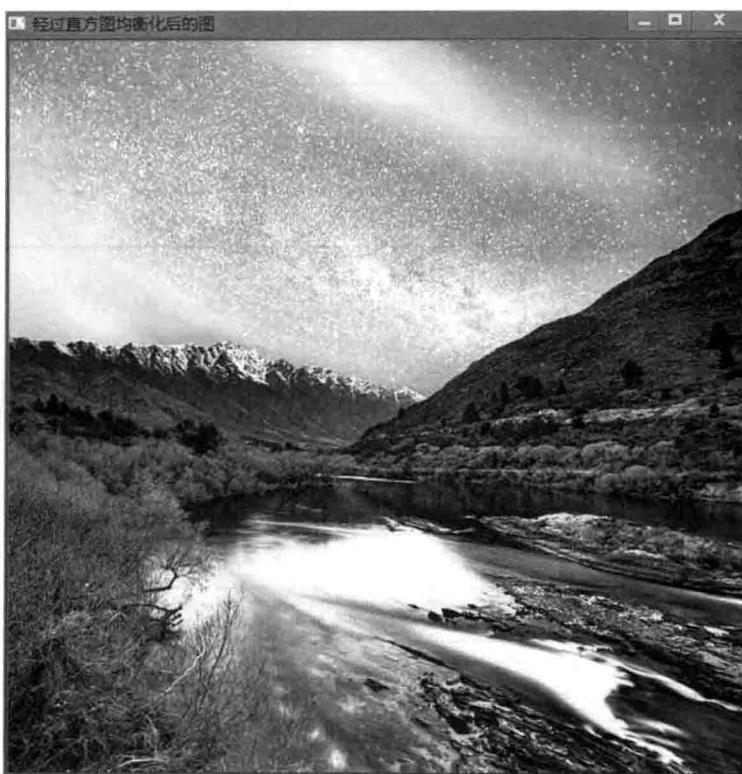


图 7.48 均衡化后的效果图

## 7.6 本章小结

在这章中我们学习了很多类型的图像变换方法。包括利用 OpenCV 进行边缘检测所用到的 canny 算子、sobel 算子，Laplace 算子以及 scharr 滤波器；进行图像特征提取的霍夫线变换、霍夫圆变换，重映射和仿射变换以及直方图均衡化。

### 本章核心函数清单

函数名称	说明	对应讲解章节
Canny	利用 Canny 算子来进行图像的边缘检测	7.1.2
Sobel	使用扩展的 Sobel 算子，来计算一阶、二阶、三阶或混合图像差分。	7.1.3
Laplacian	计算出图像经过拉普拉斯变换后的结果。	7.1.4
Scharr	使用 Scharr 滤波器运算符计算 x 或 y 方向的图像差分。	7.1.5
HoughLines	找出采用标准霍夫变换的二值图像线条	7.2.4
HoughLinesP	采用累计概率霍夫变换 (PPHT) 来找出二值图像中的直线	7.2.5
HoughCircles	利用霍夫变换算法检测出灰度图中的圆	7.2.8
remap	根据指定的映射形式，将源图像进行重映射几何变换	7.3.2
warpAffine	依据公式对图像做仿射变换	7.4.3
getRotationMatrix2D	计算二维旋转变换矩阵	7.4.4
equalizeHist	实现图像的直方图均衡化	7.5.2

### 本章示例程序清单

示例程序序号	程序说明	对应章节
56	Canny 边缘检测	7.1.2
57	Sobel 算子的使用	7.1.3
58	Laplacian 算子的使用	7.1.4
59	Scharr 滤波器	7.1.5
60	综合示例：边缘检测	7.1.6
61	标准霍夫变换：HoughLines()函数的使用	7.2.4
62	累计概率霍夫变换：HoughLinesP()函数	7.2.5

续表

示例程序序号	程序说明	对应章节
63	霍夫圆变换： HoughCircles()函数	7.2.8
64	综合示例： 霍夫变换	7.2.9
65	实现重映射： remap()函数	7.3.3
66	综合示例程序： 实现多种重映射	7.3.4
67	仿射变换	7.4.5
68	直方图均衡化	7.5.3

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

# 第 8 章

## 图像轮廓与图像分割修复

### 导读

---

虽然 Canny 之类的边缘检测算法可以根据像素之间的差异，检测出轮廓边界的像素，但是它并没有将轮廓作为一个整体。所以，下一步便是把这些边缘像素组装成轮廓。

在第 6 章介绍图像处理时，我们已经讨论了几个图像分割的算法，如图像形态学、阈值化以及金字塔分割法。而本章将进一步介绍分水岭算法，以及如何进行图像修补。

### 本章你将学到：

---

- 如何查找并绘制轮廓
- 如何寻找物体的凸包
- 如何使用多边形逼近物体
- 认识图像的矩
- 如何利用 OpenCV 进行图像修补

## 8.1 查找并绘制轮廓

一个轮廓一般对应一系列的点，也就是图像中的一条曲线。其表示方法可能根据不同的情况而有所不同。在 OpenCV 中，可以用 `findContours()` 函数从二值图像中查找轮廓。

### 8.1.1 寻找轮廓：`findContours()` 函数

`findContours()` 函数用于在二值图像中寻找轮廓。

```
C++: void findContours(InputOutputArray image, OutputArrayOfArrays
contours, OutputArray hierarchy, int mode, int method, Point
offset=Point())
```

- 第一个参数，`InputArray` 类型的 `image`，输入图像，即源图像，填 `Mat` 类的对象即可，且需为 8 位单通道图像。图像的非零像素被视为 1，0 像素值被保留为 0，所以图像为二进制。我们可以使用 `compare()`、`inrange()`、`threshold()`、`adaptiveThreshold()`、`canny()` 等函数由灰度图或彩色图创建二进制图像。此函数会在提取图像轮廓的同时修改图像的内容。
- 第二个参数，`OutputArrayOfArrays` 类型的 `contours`、检测到的轮廓、函数调用后的运算结果存在这里。每个轮廓存储为一个点向量，即用 `point` 类型的 `vector` 表示。
- 第三个参数，`OutputArray` 类型的 `hierarchy`，可选的输出向量，包含图像的拓扑信息。其作为轮廓数量的表示，包含了许多元素。每个轮廓 `contours[i]` 对应 4 个 `hierarchy` 元素 `hierarchy[i][0]~hierarchy[i][3]`，分别表示后一个轮廓、前一个轮廓、父轮廓、内嵌轮廓的索引编号。如果没有对应项，对应的 `hierarchy[i]` 值设置为负数。
- 第四个参数，`int` 类型的 `mode`，轮廓检索模式，取值如表 8.1 所示。

表 8.1 `findContours` 函数可选的轮廓检索模式

标识符	含义
<code>RETR_EXTERNAL</code>	表示只检测最外层轮廓。对所有轮廓，设置 <code>hierarchy[i][2]=hierarchy[i][3]=-1</code>
<code>RETR_LIST</code>	提取所有轮廓，并且放置在 <code>list</code> 中。检测的轮廓不建立等级关系
<code>RETR_CCOMP</code>	提取所有轮廓，并且将其组织为双层结构 (two-level hierarchy: 顶层为连通域的外围边界，次层为孔的内层边界)
<code>RETR_TREE</code>	提取所有轮廓，并重新建立网状的轮廓结构

- 第五个参数，`int` 类型的 `method`，为轮廓的近似办法，取值如表 8.2 所示。

表 8.2 findContours 函数可选的轮廓近似办法

标识符	含义
CHAIN_APPROX_NONE	获取每个轮廓的每个像素，相邻的两个点的像素位置差不超过 1，即 $\max(\text{abs}(x_1-x_2), \text{abs}(y_2-y_1))=1$
CHAIN_APPROX_SIMPLE	压缩水平方向，垂直方向，对角线方向的元素，只保留该方向的终点坐标，例如一个矩形轮廓只需 4 个点来保存轮廓信息
CHAIN_APPROX_TC89_L1 ， CHAIN_APPROX_TC89_KCOS	使用 Teh-Chinl 链逼近算法中的一个



同样地，在表 8.1 和 8.2 中列出的宏之前加上“CV\_”前缀，便是 OpenCV2 中可以使用的宏。如“RETR\_CCOMP”宏的 OpenCV2 版为“CV\_RETR\_CCOMP”。

- 第六个参数，Point 类型的 offset，每个轮廓点的可选偏移量，有默认值 Point()。对 ROI 图像中找出的轮廓，并要在整个图像中进行分析时，这个参数便可排上用场。

findContours 经常与 drawContours 配合使用—使用用 findContours()函数检测到图像的轮廓后，便可以用 drawContours()函数将检测到的轮廓绘制出来。接下来，让我们一起看看 drawContours()函数的用法。

下面是一个调用小示例。

```
vector<vector<Point>> contours;
findContours(image,
contours, //轮廓数组
CV_RETR_EXTERNAL, //获取外轮廓
CV_CHAIN_APPROX_NONE); // 获取每个轮廓的每个像素
```

### 8.1.2 绘制轮廓：drawContours()函数

drawContours()函数用于在图像中绘制外部或内部轮廓。

```
C++: void drawContours(InputOutputArray image, InputArrayOfArrays
contours, int contourIdx, const Scalar& color, int thickness=1, int
lineType=8, InputArray hierarchy=noArray(), int maxLevel=INT_MAX, Point
offset=Point() )
```

- 第一个参数，InputArray 类型的 image，目标图像，填 Mat 类的对象即可。
- 第二个参数，InputArrayOfArrays 类型的 contours，所有的输入轮廓。每个轮廓存储为一个点向量，即用 point 类型的 vector 表示。
- 第三个参数，int 类型的 contourIdx，轮廓绘制的指示变量。如果其为负值，则绘制所有轮廓。
- 第四个参数，const Scalar&类型的 color，轮廓的颜色。
- 第五个参数，int thickness，轮廓线条的粗细度，有默认值 1。如果其为负值（如 thickness= cv\_filled），便会绘制在轮廓的内部。可选为 FILLED 宏

(OpenCV2 版为 CV\_FILLED)。

- 第六个参数，int 类型的 lineType，线条的类型，有默认值 8。取值类型如表 8.3 所示。

表 8.3 可选线性

lineType 线性	含义
8 (默认值)	8 连通线型
4	4 连通线型
LINE_AA(OpenCV2 版为 CV_AA)	抗锯齿线型

- 第七个参数，InputArray 类型的 hierarchy，可选的层次结构信息，有默认值 noArray()。
- 第八个参数，int 类型的 maxLevel，表示用于绘制轮廓的最大等级，有默认值 INT\_MAX
- 第九个参数，Point 类型的 offset，可选的轮廓偏移参数，用指定的偏移量 offset= (dx, dy) 偏移需要绘制的轮廓，有默认值 Point()。

下面是一个调用小示例。

```
//在白色图像上绘制黑色轮廓
Mat result(image.size(), CV_8U, cv::Scalar(255));
drawContours(result, contours,
-1, // 绘制所有轮廓
Scalar(0), // 绘制颜色取黑色
3); // 轮廓线的线宽为 3
```

### 8.1.3 基础示例程序：轮廓查找

讲解完这两个参数，让我们看一个小型的完整示例程序，看看在 OpenCV 实际运用中，这两个函数的组合用法。此程序详细注释的代码如下。

```
#include <opencv2/opencv.hpp>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;

int main( int argc, char** argv )
{
    // 【1】载入原始图，且必须以二值图模式载入
    Mat srcImage = imread("1.jpg", 0);
    imshow("原始图", srcImage);

    // 【2】初始化结果图
    Mat dstImage = Mat::zeros(srcImage.rows, srcImage.cols, CV_8UC3);

    // 【3】srcImage 取大于阈值 119 的那部分
    srcImage = srcImage > 119;
    imshow("取阈值后的原始图", srcImage);
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
//【4】定义轮廓和层次结构
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;

//【5】查找轮廓
//此句代码的OpenCV2版为：
//findContours( srcImage, contours, hierarchy, CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE );
//此句代码的OpenCV3版为：
findContours( srcImage, contours, hierarchy, RETR_CCOMP,
CHAIN_APPROX_SIMPLE );

//【6】遍历所有顶层的轮廓，以随机颜色绘制出每个连接组件颜色
int index = 0;
for( ; index >= 0; index = hierarchy[index][0] )
{
    Scalar color( rand()&255, rand()&255, rand()&255 );
    //此句代码的OpenCV2版为：
    //drawContours( dstImage, contours, index, color, CV_FILLED, 8,
hierarchy );
    //此句代码的OpenCV3版为：
    drawContours( dstImage, contours, index, color, FILLED, 8,
hierarchy );
}

//【7】显示最后的轮廓图
imshow( "轮廓图", dstImage );

waitKey(0);
}
```

程序运行截图如图 8.1、8.2、8.3 所示。



图 8.1 原始图



图 8.2 取阈值后的原始图

图 8.3 轮廓图

#### 8.1.4 综合示例程序：查找并绘制轮廓

除了上述这个精简版的示例程序，还为大家准备了一个更加复杂一些的关于查找并绘制轮廓的综合示例程序。此程序利用了图像平滑技术（blur()函数）和边缘检测技术（canny()函数），根据滑动条的调节，可以动态地检测出图形的轮廓。

程序经过详细注释的代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
-----
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;

-----【宏定义部分】-
//      描述：定义一些辅助宏
-----
#define WINDOW_NAME1 "【原始图窗口】"          //为窗口标题定义的宏
#define WINDOW_NAME2 "【轮廓图】"                //为窗口标题定义的宏
```

```
-----【全局变量声明部分】-----
//      描述：全局变量的声明
-----
Mat g_srcImage;
Mat g_grayImage;
int g_nThresh = 80;
int g_nThresh_max = 255;
RNG g_rng(12345);
Mat g_cannyMat_output;
vector<vector<Point>> g_vContours;
vector<Vec4i> g_vHierarchy;

-----【全局函数声明部分】-----
//      描述：全局函数的声明
-----
static void ShowHelpText();
void on_ThresholdChange(int, void* );

-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
-----
int main( int argc, char** argv )
{
    //【0】改变 console 字体颜色
    system("color 1F");

    //【0】显示欢迎和帮助文字
    ShowHelpText();

    // 加载源图像
    g_srcImage = imread( "1.jpg", 1 );
    if(!g_srcImage.data)
    {
        printf("读取图片错误，请确定目录下是否有 imread 函数指定的图片存在! \n");
        return false;
    }

    // 转成灰度并模糊化降噪
    cvtColor( g_srcImage, g_grayImage, COLOR_BGR2GRAY );
    blur( g_grayImage, g_grayImage, Size(3,3) );

    // 创建窗口
    namedWindow( WINDOW_NAME1, WINDOW_AUTOSIZE );
    imshow( WINDOW_NAME1, g_srcImage );

    //创建滚动条并初始化
    createTrackbar( "canny 阈值", WINDOW_NAME1, &g_nThresh,
                    g_nThresh_max, on_ThresholdChange );
    on_ThresholdChange( 0, 0 );
```

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

//-----【on_ThresholdChange() 函数】-----
//      描述：回调函数
//-----
void on_ThresholdChange(int, void*)
{
    // 用 Canny 算子检测边缘
    Canny(g_grayImage, g_cannyMat_output, g_nThresh, g_nThresh*2, 3);

    // 寻找轮廓
    findContours(g_cannyMat_output, g_vContours, g_vHierarchy,
    RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0) );

    // 绘出轮廓
    Mat drawing = Mat::zeros(g_cannyMat_output.size(), CV_8UC3 );
    for( int i = 0; i< g_vContours.size(); i++ )
    {
        Scalar color = Scalar( g_rng.uniform(0, 255),
        g_rng.uniform(0,255), g_rng.uniform(0,255) );//任意值
        drawContours(drawing, g_vContours, i, color, 2, 8, g_vHierarchy,
        0, Point() );
    }

    // 显示效果图
    imshow(WINDOW_NAME2, drawing );
}

//-----【ShowHelpText() 函数】-----
//      描述：输出一些帮助信息
//-----
static void ShowHelpText()
{
    //输出欢迎信息和 OpenCV 版本
    printf("\n\n\t\t非常感谢购买《OpenCV3 编程入门》一书! \n");
    printf("\n\n\t\t此为本书 OpenCV3 版的第 70 个配套示例程序\n");
    printf("\n\t\t当前使用的 OpenCV 版本为：" CV_VERSION );
    printf("\n\n-----\n");

    //输出一些帮助信息
    printf(" \n\n\t 欢迎来到【在图形中寻找轮廓】示例程序~\n\n");
    printf(" \n\t按键操作说明： \n\n"
    "\t\t 键盘按键任意键- 退出程序\n\n"
    "\t\t 滑动滚动条-改变阈值\n" );
}
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

可以在原始图窗口中通过滑动条调节阈值，程序便会根据不同的阈值得到不同的轮廓图。运行效果图如图 8.4~8.8 所示。

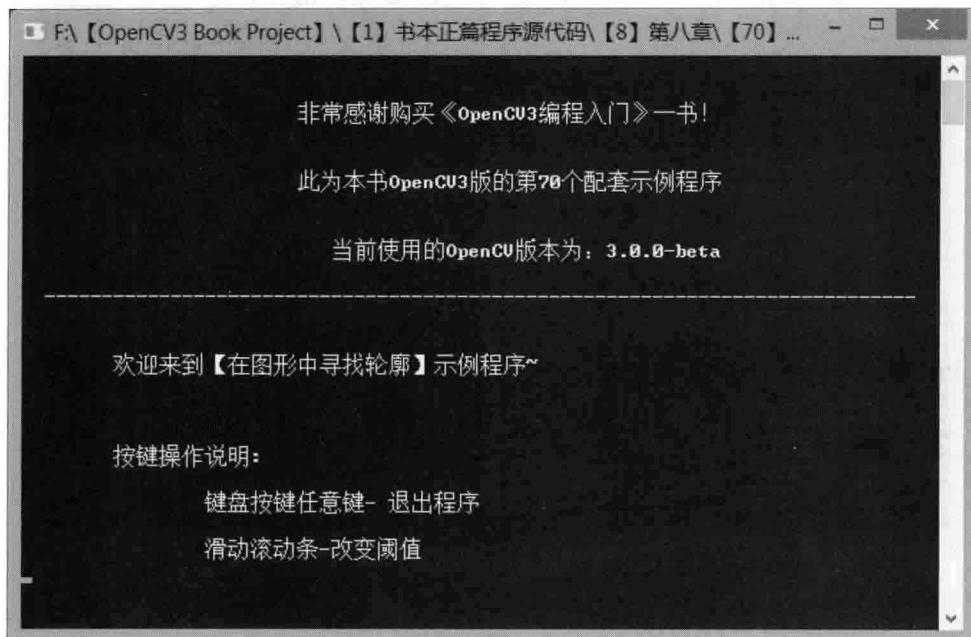


图 8.4 运行帮助文字截图



图 8.5 原始图窗口

图 8.6 阈值为 0 时的轮廓图

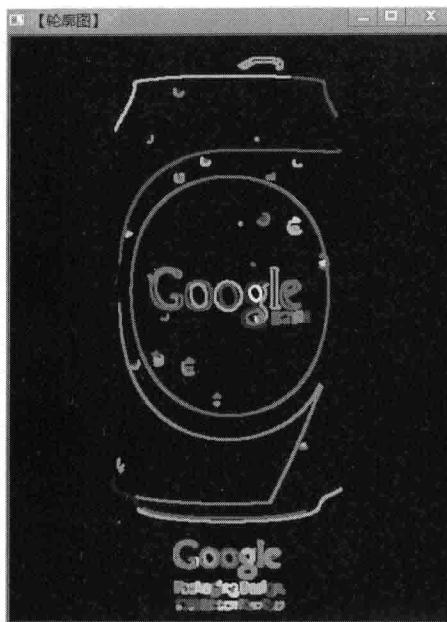


图 8.7 阈值为 80 时的轮廓图

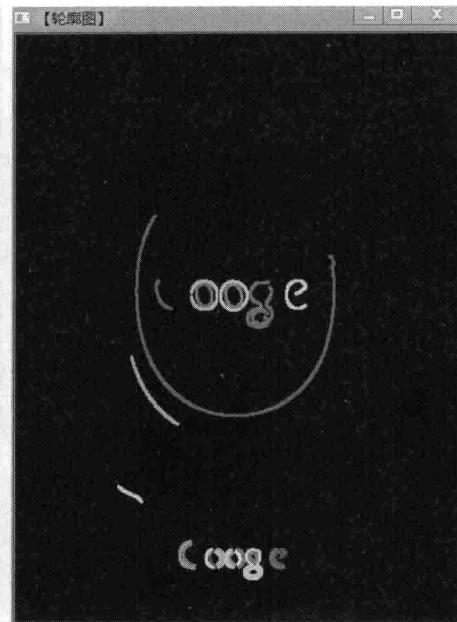


图 8.8 阈值为 187 时的轮廓图

## 8.2 寻找物体的凸包

### 8.2.1 凸包

凸包（Convex Hull）是一个计算几何（图形学）中常见的概念。简单来说，给定二维平面上的点集，凸包就是将最外层的点连接起来构成的凸多边型，它是能包含点集中所有点的。理解物体形状或轮廓的一种比较有用的方法便是计算一个物体的凸包，然后计算其凸缺陷（convexity defects）。很多复杂物体的特性能很好地被这种缺陷表现出来。

如图 8.9 所示，我们用人手图来举例说明凸缺陷这一概念。手周围深色的线描画出了凸包，A 到 H 被标出的区域是凸包的各个“缺陷”。正如看到的，这些凸度缺陷提供了手以及手状态的特征表现的方法。

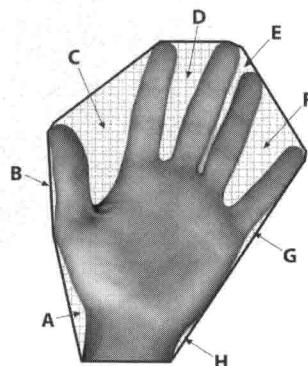


图 8.9 手掌凸缺陷示例

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

新版 OpenCV 中，convexHull 函数用于寻找图像点集中的凸包，我们一起来看一下这个函数。

### 8.2.2 寻找凸包：convexHull()函数

上文已经提到过，convexHull()函数用于寻找图像点集中的凸包，其原型声明如下。

```
C++: void convexHull(InputArray points, OutputArray hull, bool  
clockwise=false, bool returnPoints=true )
```

- 第一个参数，InputArray 类型的 points，输入的二维点集，可以填 Mat 类型或者 std::vector。
- 第二个参数，OutputArray 类型的 hull，输出参数，函数调用后找到的凸包。
- 第三个参数，bool 类型的 clockwise，操作方向标识符。当此标识符为真时，输出的凸包为顺时针方向。否则，就为逆时针方向。并且是假定坐标系的 x 轴指向右，y 轴指向上方。
- 第四个参数，bool 类型的 returnPoints，操作标志符，默认值 true。当标志符为真时，函数返回各凸包的各个点。否则，它返回凸包各点的指数。当输出数组是 std::vector 时，此标志被忽略。

### 8.2.3 基础示例程序：凸包检测基础

为了理解凸包检测的运用方法，下面放出一个完整的示例程序。程序中会首先随机生成 3~103 个坐标值随机的彩色点，然后利用 convexHull，对由这些点链接起来的图形求凸包。

操作说明如图 8.10 所示。

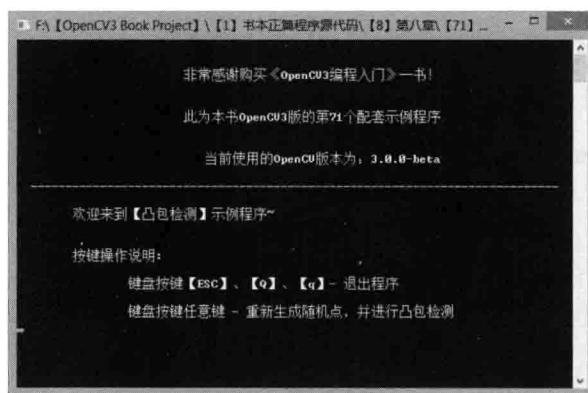


图 8.10 操作说明

此程序详细注释的程序源代码如下。

```
#include "opencv2/imgproc/imgproc.hpp"  
#include "opencv2/highgui/highgui.hpp"  
#include <iostream>  
using namespace cv;
```

```
-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
-----
int main()
{
    //初始化变量和随机值
    Mat image(600, 600, CV_8UC3);
    RNG& rng = theRNG();

    //循环，按下 ESC,Q,q 键程序退出，否则有键按下便一直更新
    while(1)
    {
        //参数初始化
        char key;//键值
        int count = (unsigned)rng%100 + 3;//随机生成点的数量
        vector<Point> points; //点值

        //随机生成点坐标
        for(int i = 0; i < count; i++ )
        {
            Point point;
            point.x = rng.uniform(image.cols/4, image.cols*3/4);
            point.y = rng.uniform(image.rows/4, image.rows*3/4);

            points.push_back(point);
        }

        //检测凸包
        vector<int> hull;
        convexHull(Mat(points), hull, true);

        //绘制出随机颜色的点
        image = Scalar::all(0);
        for(int i = 0; i < count; i++ )
            circle(image, points[i], 3, Scalar(rng.uniform(0, 255),
rng.uniform(0, 255), rng.uniform(0, 255)), FILLED, LINE_AA);

        //准备参数
        int hullcount = (int)hull.size();//凸包的边数
        Point point0 = points[hull[hullcount-1]];//连接凸包边的坐标点

        //绘制凸包的边
        for(int i = 0; i < hullcount; i++ )
        {
            Point point = points[hull[i]];
            line(image, point0, point, Scalar(255, 255, 255), 2, LINE_AA);
            point0 = point;
        }

        //显示效果图
    }
}
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

imshow("凸包检测示例", image);

//按下 ESC,Q,或者 q, 程序退出
key = (char)waitKey();
if( key == 27 || key == 'q' || key == 'Q' )
    break;
}

return 0;
}

```

程序的运行截图如图 8.11、8.12 所示。

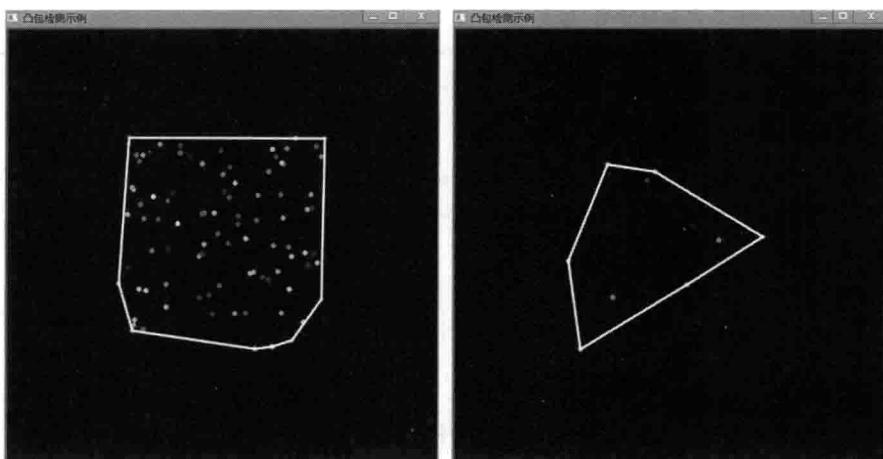


图 8.11 运行截图 (1)

图 8.12 运行截图 (2)

#### 8.2.4 综合示例程序：寻找和绘制物体的凸包

这一节的综合示例程序，依然是结合滑动条，通过滑动条控制阈值，来得到不同的凸包检测效果图。程序详细注释的源代码如下。

```

-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;

-----【宏定义部分】-----
//  描述：定义一些辅助宏
//

#define WINDOW_NAME1 "【原始图窗口】"           //为窗口标题定义
的宏
#define WINDOW_NAME2 "【效果图窗口】"           //为窗口标题定义
的宏

-----【全局变量声明部分】-----

```

```
// 描述：全局变量的声明
//-----
Mat g_srcImage; Mat g_grayImage;
int g_nThresh = 50;
int g_maxThresh = 255;
RNG g_rng(12345);
Mat srcImage_copy = g_srcImage.clone();
Mat g_thresholdImage_output;
vector<vector<Point>> g_vContours;
vector<Vec4i> g_vHierarchy;

//-----【全局函数声明部分】-----
// 描述：全局函数的声明
//-----
static void ShowHelpText();
void on_ThresholdChange(int, void* );

//-----【main()函数】-----
// 描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----
int main()
{
    // 加载源图像
    g_srcImage = imread( "1.jpg", 1 );

    // 将原图转换成灰度图并进行模糊降噪
    cvtColor( g_srcImage, g_grayImage, COLOR_BGR2GRAY );
    blur( g_grayImage, g_grayImage, Size(3,3) );

    // 创建原图窗口并显示
    namedWindow( WINDOW_NAME1, WINDOW_AUTOSIZE );
    imshow( WINDOW_NAME1, g_srcImage );

    // 创建滚动条
    createTrackbar( " 阈值:", WINDOW_NAME1, &g_nThresh, g_maxThresh,
on_ThresholdChange );
    on_ThresholdChange( 0, 0 );//调用一次进行初始化

    waitKey(0);
    return(0);
}

//-----【 thresh_callback() 函数】-----
// 描述：回调函数
//-----
void on_ThresholdChange(int, void* )
{
    // 对图像进行二值化，控制阈值
    threshold( g_grayImage, g_thresholdImage_output, g_nThresh, 255,
THRESH_BINARY );

    // 寻找轮廓
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
findContours( g_thresholdImage_output, g_vContours, g_vHierarchy,
RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0) );

// 遍历每个轮廓，寻找其凸包
vector<vector<Point>> hull( g_vContours.size() );
for( unsigned int i = 0; i < g_vContours.size(); i++ )
{
    convexHull( Mat(g_vContours[i]), hull[i], false );
}

// 绘出轮廓及其凸包
Mat drawing = Mat::zeros( g_thresholdImage_output.size(), CV_8UC3 );
for(unsigned int i = 0; i < g_vContours.size(); i++ )
{
    Scalar color = Scalar( g_rng.uniform(0, 255),
g_rng.uniform(0,255), g_rng.uniform(0,255) );
    drawContours( drawing, g_vContours, i, color, 1, 8,
vector<Vec4i>(), 0, Point() );
    drawContours( drawing, hull, i, color, 1, 8, vector<Vec4i>(), 0,
Point() );
}

// 显示效果图
imshow( WINDOW_NAME2, drawing );
}
```

程序运行后，可以在原始图窗口中调节阈值，改变全局变量 g\_nThresh 的值，以改变 g\_thresholdImage\_output 参数，最后 findContours 以此参数为输入，查找不同的轮廓图。经过一系列的处理，最后得到的凸包图形便有精细度上的差异。运行效果如图 8.13~8.16 所示。



图 8.13 原始图窗口



图 8.14 阈值为 50 时的效果图



图 8.15 阈值为 113 时的效果图



图 8.16 阈值为 164 时的效果图

### 8.3 使用多边形将轮廓包围

在实际应用中，常常会有将检测到的轮廓用多边形表示出来的需求。本节就为大家讲解如何用多边形来表示出轮廓，或者说如何根据轮廓提取出多边形。先让我们一起学习用 OpenCV 创建包围轮廓的多边形边界时会接触到的一些函数。

#### 8.3.1 返回外部矩形边界：`boundingRect()`函数

此函数计算并返回指定点集最外面（up-right）的矩形边界。

C++: `Rect boundingRect(InputArray points)`

其唯一的一个参数为输入的二维点集，可以是 `std::vector` 或 `Mat` 类型。

#### 8.3.2 寻找最小包围矩形：`minAreaRect()`函数

此函数用于对给定的 2D 点集，寻找可旋转的最小面积的包围矩形。

C++: `RotatedRect minAreaRect(InputArray points)`

其唯一的一个参数为输入的二维点集，可以为 `std::vector` 或 `Mat` 类型。

#### 8.3.3 寻找最小包围圆形：`minEnclosingCircle()`函数

`minEnclosingCircle` 函数的功能是利用一种迭代算法，对给定的 2D 点集，去寻找面积最小的可包围它们的圆形。

C++: `void minEnclosingCircle(InputArray points, Point2f& center, float& radius)`

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

- 第一个参数, InputArray 类型的 points, 输入的二维点集, 可以为 std::vector<> 或 Mat 类型。
- 第二个参数, Point2f&类型的 center, 圆的输出圆心。
- 第三个参数, float&类型的 radius, 圆的输出半径。

### 8.3.4 用椭圆拟合二维点集: fitEllipse()函数

此函数的作用是用椭圆拟合二维点集。

C++: RotatedRect fitEllipse(InputArray points)

其唯一的一个参数为输入的二维点集, 可以为 std::vector<>或 Mat 类型。

### 8.3.5 逼近多边形曲线: approxPolyDP()函数

approxPolyDP 函数的作用是用指定精度逼近多边形曲线。

C++: void approxPolyDP(InputArray curve, OutputArray approxCurve, double epsilon, bool closed)

- 第一个参数, InputArray 类型的 curve, 输入的二维点集, 可以为 std::vector<> 或 Mat 类型。
- 第二个参数, OutputArray 类型的 approxCurve, 多边形逼近的结果, 其类型应该和输入的二维点集的类型一致。
- 第三个参数, double 类型的 epsilon, 逼近的精度, 为原始曲线和近似曲线间的最大值。
- 第四个参数, bool 类型的 closed, 如果其为真, 则近似的曲线为封闭曲线(第一个顶点和最后一个顶点相连), 否则, 近似的曲线不封闭。

讲解完上述的几个函数, 下面我们来通过完整的示例程序理解所学。笔者为大家准备了两个示例程序, 分别为用矩形和圆形边界包围生成的轮廓。

### 8.3.6 基础示例程序: 创建包围轮廓的矩形边界

此示例程序以 minAreaRect 函数为核心, 先随机生成 3~103 个彩色点, 然后绘制一个可以旋转的最小的矩形, 将这些点全部包含进去。操作依然是通过键盘按下任意键来重新生成随机点, 并寻找最小面积的包围矩形。

详细注释的程序代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述: 包含程序所依赖的头文件和命名空间
//-----  
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;
using namespace std;
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
int main()
{
    //初始化变量和随机值
    Mat image(600, 600, CV_8UC3);
    RNG& rng = theRNG();

    //循环，按下 ESC,Q,q 键程序退出，否则有键按下便一直更新
    while(1)
    {
        //参数初始化
        int count = rng.uniform(3, 103); //随机生成点的数量
        vector<Point> points; //点值

        //随机生成点坐标
        for(int i = 0; i < count; i++)
        {

            Point point;
            point.x = rng.uniform(image.cols/4, image.cols*3/4);
            point.y = rng.uniform(image.rows/4, image.rows*3/4);

            points.push_back(point);
        }

        //对给定的 2D 点集，寻找最小面积的包围矩形
        RotatedRect box = minAreaRect(Mat(points));
        Point2f vertex[4];
        box.points(vertex);

        //绘制出随机颜色的点
        image = Scalar::all(0);
        for( int i = 0; i < count; i++ )
            circle( image, points[i], 3, Scalar(rng.uniform(0, 255),
rng.uniform(0, 255), rng.uniform(0, 255)), FILLED, LINE_AA );

        //绘制出最小面积的包围矩形
        for( int i = 0; i < 4; i++ )
            line(image, vertex[i], vertex[(i+1)%4], Scalar(100, 200, 211),
2, LINE_AA);

        //显示窗口
        imshow( "矩形包围示例", image );

        //按下 ESC,Q,或者 q, 程序退出
        char key = (char)waitKey();
        if( key == 27 || key == 'q' || key == 'Q' ) // 'ESC'
            break;
    }

    return 0;
}
```

程序注释已经十分详细。我们运行程序，便可以看到随机生成的彩色的点，以及包围着它们的矩形。按下除【ESC】、【Q】、【q】之外的任意键，便可以得到新的彩色的点和矩形。如图 8.17、8.18 所示。

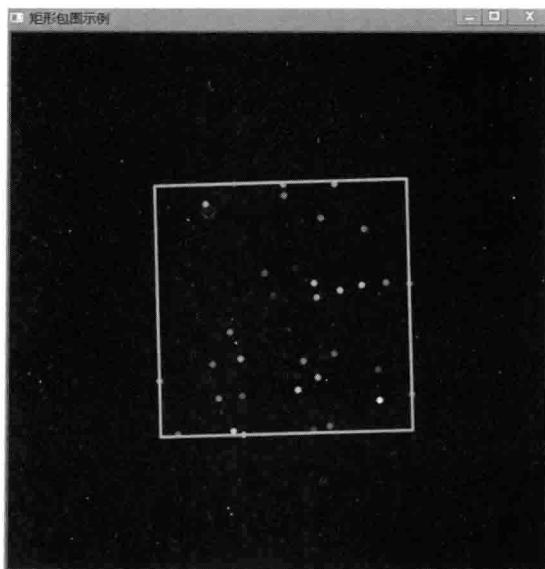


图 8.17 程序截图 (1)

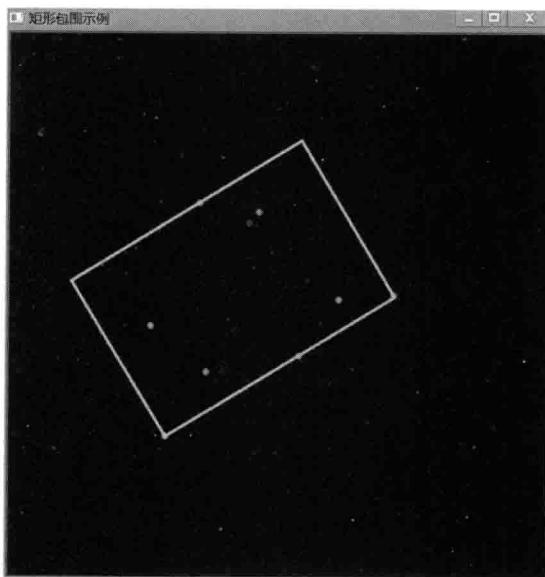


图 8.18 程序截图 (2)

### 8.3.7 基础示例程序：创建包围轮廓的圆形边界

本小节给出的程序和上一小节的程序类似，只不过核心函数替换为 `minEnclosingCircle`，依然是先随机生成 3~103 个彩色点，不同的是绘制的是一个圆而不是矩形，将这些点组成的轮廓包含进去。为了大家学习方便，这里依然是贴出此程序详细注释的源代码。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;
using namespace std;

int main()
{
    //初始化变量和随机值
    Mat image(600, 600, CV_8UC3);
    RNG& rng = theRNG();

    //循环，按下 ESC,Q,q 键程序退出，否则有键按下便一直更新
    while(1)
    {
        //参数初始化
        int count = rng.uniform(3, 103); //随机生成点的数量
        vector<Point> points; //点值

        //随机生成点坐标
        for(int i = 0; i < count; i++ )
        {

            Point point;
            point.x = rng.uniform(image.cols/4, image.cols*3/4);
            point.y = rng.uniform(image.rows/4, image.rows*3/4);

            points.push_back(point);
        }

        //对给定的 2D 点集，寻找最小面积的包围圆
        Point2f center;
        float radius = 0;
        minEnclosingCircle(Mat(points), center, radius);

        //绘制出随机颜色的点
        image = Scalar::all(0);
        for( int i = 0; i < count; i++ )
            circle( image, points[i], 3, Scalar(rng.uniform(0, 255),
rng.uniform(0, 255), rng.uniform(0, 255)), FILLED, LINE_AA );

        //绘制出最小面积的包围圆
        circle(image, center, cvRound(radius), Scalar(rng.uniform(0,
255), rng.uniform(0, 255), rng.uniform(0, 255)), 2, LINE_AA);

        //显示窗口
        imshow( "圆形包围示例", image );
    }
}
```

```
//按下 ESC, Q, 或者 q, 程序退出
char key = (char)waitKey();
if( key == 27 || key == 'q' || key == 'Q' ) // 'ESC'
    break;
}

return 0;
}
```

程序注释依然是十分详细。我们运行程序，便可以看到随机生成的彩色的点，以及包围着它们的圆。按下除【ESC】、【Q】、【q】之外的任意键，便可以得到新的彩色的点和圆。如图 8.19、8.20 所示。



图 8.19 程序截图（1）



图 8.20 程序截图（2）

### 8.3.8 综合示例程序：使用多边形包围轮廓

经过上述两个基础示例程序的学习，相信大家应该对 minAreaRect 和 minEnclosingCircle 函数的用法有了一定的认识。这两个示例程序中，处理的轮廓都是程序自己随机生成的点。在接下来这个综合一点的示例程序中，让我们载入一幅图像，用上文中学到的函数来创建包围轮廓的矩形和圆形边界框。

程序详细注释的代码如下。

```
-----【头文件、命名空间包含部分】-----
// 描述：包含程序所依赖的头文件和命名空间
-----
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;
using namespace std;

-----【宏定义部分】-----
// 描述：定义一些辅助宏
-----
#define WINDOW_NAME1 "[原始图窗口]"           //为窗口标题定义的宏
#define WINDOW_NAME2 "[效果图窗口]"           //为窗口标题定义的宏

-----【全局变量声明部分】-----
// 描述：全局变量的声明
-----
Mat g_srcImage;
Mat g_grayImage;
int g_nThresh = 50; //阈值
int g_nMaxThresh = 255; //阈值最大值
RNG g_rng(12345); //随机数生成器

-----【全局函数声明部分】-----
// 描述：全局函数的声明
-----
void onContoursChange(int, void* );
static void ShowHelpText();

-----【main()函数】-----
// 描述：控制台应用程序的入口函数，我们的程序从这里开始执行
-----
int main()
{
    //【0】改变 console 字体颜色
    system("color 1A");

    //【1】载入 3 通道的原图像
    g_srcImage = imread("1.jpg", 1);
    if(!g_srcImage.data) { printf("读取图片错误，请确定目录下是否有 imread\n函数指定的图片存在~! \n"); return false; }
```

```
//【2】得到原图的灰度图像并进行平滑
cvtColor( g_srcImage, g_grayImage, COLOR_BGR2GRAY );
blur( g_grayImage, g_grayImage, Size(3,3) );

//【3】创建原始图窗口并显示
namedWindow( WINDOW_NAME1, WINDOW_AUTOSIZE );
imshow( WINDOW_NAME1, g_srcImage );

//【4】设置滚动条并调用一次回调函数
createTrackbar( " 阈值:", WINDOW_NAME1, &g_nThresh, g_nMaxThresh,
on_ContoursChange );
on_ContoursChange( 0, 0 );

waitKey(0);

return(0);
}

//-----【on_ContoursChange() 函数】-----
//      描述：回调函数
//-----
void on_ContoursChange(int, void* )
{
    //定义一些参数
    Mat threshold_output;
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;

    // 使用 Threshold 检测边缘
    threshold( g_grayImage, threshold_output, g_nThresh, 255,
THRESH_BINARY );

    // 找出轮廓
    findContours( threshold_output, contours, hierarchy, RETR_TREE,
CHAIN_APPROX_SIMPLE, Point(0, 0) );

    // 多边形逼近轮廓 + 获取矩形和圆形边界框
    vector<vector<Point>> contours_poly( contours.size() );
    vector<Rect> boundRect( contours.size() );
    vector<Point2f>center( contours.size() );
    vector<float>radius( contours.size() );

    //一个循环，遍历所有部分，进行本程序最核心的操作
    for( unsigned int i = 0; i < contours.size(); i++ )
    {
        approxPolyDP( Mat(contours[i]), contours_poly[i], 3, true );//用指定精度逼近多边形曲线
        boundRect[i] = boundingRect( Mat(contours_poly[i]) );//计算点集的最外面 (up-right) 矩形边界
        minEnclosingCircle( contours_poly[i], center[i], radius[i] );//对给定的 2D 点集，寻找最小面积的包围圆形
    }
}
```

```
// 绘制多边形轮廓 + 包围的矩形框 + 圆形框
Mat drawing = Mat::zeros( threshold_output.size(), CV_8UC3 );
for( int unsigned i = 0; i<contours.size(); i++ )
{
    Scalar color = Scalar( g_rng.uniform(0, 255),
g_rng.uniform(0,255), g_rng.uniform(0,255) );//随机设置颜色
    drawContours( drawing, contours_poly, i, color, 1, 8,
vector<Vec4i>(), 0, Point() );//绘制轮廓
    rectangle( drawing, boundRect[i].tl(), boundRect[i].br(), color,
2, 8, 0 );//绘制矩形
    circle( drawing, center[i], (int)radius[i], color, 2, 8, 0 );//绘制圆
}

// 显示效果图窗口
namedWindow( WINDOW_NAME2, WINDOW_AUTOSIZE );
imshow( WINDOW_NAME2, drawing );
}
```

运行此程序。会得到两个窗口，一个带滑动条的原始图窗口，和用于显示创建了包围轮廓的矩形和圆形边界框的效果图窗口。我们可以通过调节滑动条来改变阈值，以得到不同的效果图。如图 8.21~8.24 所示。

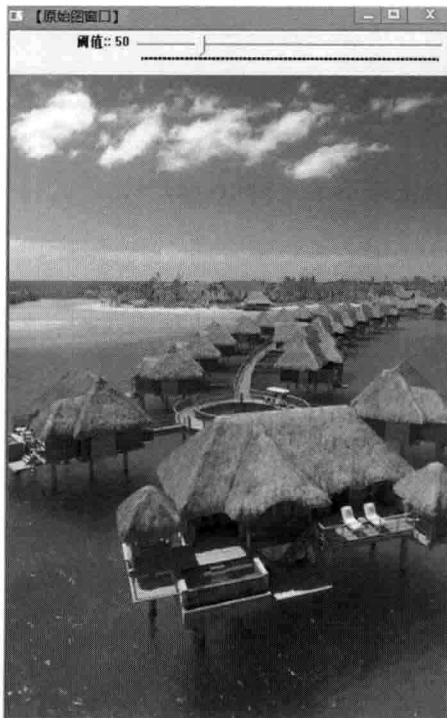


图 8.21 原始图

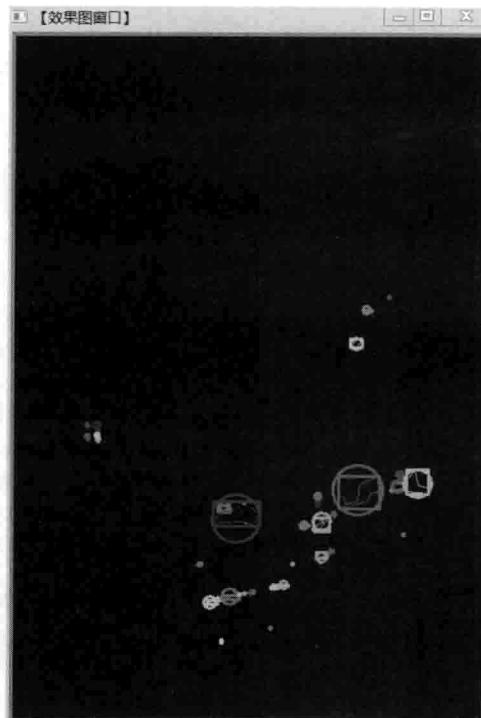


图 8.22 阈值为 21 时的效果图

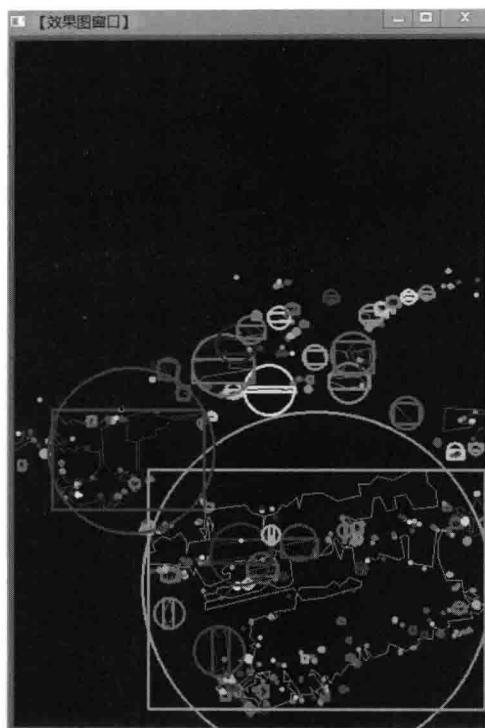


图 8.23 阈值为 55 时的效果图



图 8.24 阈值为 103 时的效果图

## 8.4 图像的矩

矩函数在图像分析中有着广泛的应用，如模式识别、目标分类、目标识别与方位估计、图像编码与重构等。一个从一幅数字图形中计算出来的矩集，通常描述了该图像形状的全局特征，并提供了大量的关于该图像不同类型的几何特性信息，比如大小、位置、方向及形状等。图像矩的这种特性描述能力被广泛地应用在各种图像处理、计算机视觉和机器人技术领域的目标识别与方位估计中。一阶矩与形状有关，二阶矩显示曲线围绕直线平均值的扩展程度，三阶矩则是关于平均值的对称性的测量。由二阶矩和三阶矩可以导出一组共 7 个不变矩。而不变矩是图像的统计特性，满足平移、伸缩、旋转均不变的不变性，在图像识别领域得到了广泛的应用。

那么，在 OpenCV 中，如何计算一个图像的矩呢？一般由 `moments`、`contourArea`、`arcLength` 这三个函数配合求取。

- 使用 `moments` 计算图像所有的矩(最高到 3 阶)
- 使用 `contourArea` 来计算轮廓面积
- 使用 `arcLength` 来计算轮廓或曲线长度

下面对其进行一一剖析。

### 8.4.1 矩的计算: moments()函数

moments()函数用于计算多边形和光栅形状的最高达三阶的所有矩。矩用来计算形状的重心、面积，主轴和其他形状特征，如 7Hu 不变量等。

```
C++: Moments moments(InputArray array, bool binaryImage=false )
```

- 第一个参数，InputArray 类型的 array，输入参数，可以是光栅图像（单通道，8 位或浮点的二维数组）或二维数组（1N 或 N1）。
- 第二个参数，bool 类型的 binaryImage，有默认值 false。若此参数取 true，则所有非零像素为 1。此参数仅对于图像使用。

需要注意的是，此参数的返回值返回运行后的结果。

### 8.4.2 计算轮廓面积: contourArea()函数

contourArea()函数用于计算整个轮廓或部分轮廓的面积

```
C++: double contourArea(InputArray contour, bool oriented=false)
```

- 第一个参数，InputArray 类型的 contour，输入的向量，二维点（轮廓顶点），可以为 std::vector 或 Mat 类型。
- 第二个参数，bool 类型的 oriented，面向区域标识符。若其为 true，该函数返回一个带符号的面积值，其正负取决于轮廓的方向（顺时针还是逆时针）。根据这个特性我们可以根据面积的符号来确定轮廓的位置。需要注意的是，这个参数有默认值 false，表示以绝对值返回，不带符号。

其调用示例如下。

```
vector<Point> contour;
contour.push_back(Point2f(0, 0));
contour.push_back(Point2f(10, 0));
contour.push_back(Point2f(10, 10));
contour.push_back(Point2f(5, 4));

double area0 = contourArea(contour);
vector<Point> approx;
approxPolyDP(contour, approx, 5, true);
double areal = contourArea(approx);

cout << "area0 =" << area0 << endl <<
    "areal =" << areal << endl <<
    "approx poly vertices" << approx.size() << endl;
```

### 8.4.3 计算轮廓长度: arcLength()函数

arcLength()函数用于计算封闭轮廓的周长或曲线的长度。

```
C++: double arcLength(InputArray curve, bool closed)
```

- 第一个参数，InputArray 类型的 curve，输入的二维点集，可以为 std::vector

或 Mat 类型。

- 第二个参数，bool 类型的 closed，一个用于指示曲线是否封闭的标识符，有默认值 closed，表示曲线封闭。

#### 8.4.4 综合示例程序：查找和绘制图像轮廓矩

学习完函数的讲解，让我们一起通过一个综合的示例程序，真正了解本节内容的实战用法。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----  
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;

-----【宏定义部分】-----
//      描述：定义一些辅助宏
//-----  
#define WINDOW_NAME1 "【原始图】"           //为窗口标题定义的宏
#define WINDOW_NAME2 "【图像轮廓】"           //为窗口标题定义的宏

-----【全局变量声明部分】-----
//      描述：全局变量的声明
//-----  
Mat g_srcImage; Mat g_grayImage;
int g_nThresh = 100;
int g_nMaxThresh = 255;
RNG g_rng(12345);
Mat g_cannyMat_output;
vector<vector<Point>> g_vContours;
vector<Vec4i> g_vHierarchy;

-----【全局变量声明部分】-----
//      描述：全局变量的声明
//-----  
void on_ThresholdChange(int, void* );
static void ShowHelpText();

-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----  
int main( int argc, char** argv )
{
    //【0】改变 console 字体颜色
    system("color 1E");

    // 读入原图像，返回 3 通道图像数据
```

```
g_srcImage = imread( "l.jpg", 1 );

// 把原图像转化成灰度图像并进行平滑
cvtColor( g_srcImage, g_grayImage, COLOR_BGR2GRAY );
blur( g_grayImage, g_grayImage, Size(3,3) );

// 创建新窗口
namedWindow( WINDOW_NAME1, WINDOW_AUTOSIZE );
imshow( WINDOW_NAME1, g_srcImage );

// 创建滚动条并进行初始化
createTrackbar( " 阈值", WINDOW_NAME1, &g_nThresh, g_nMaxThresh,
on_ThresholdChange );
on_ThresholdChange( 0, 0 );

waitKey(0);
return(0);
}

//-----【 on_ThresholdChange() 函数】-----
//      描述：回调函数
//-----
void on_ThresholdChange(int, void*)
{
    // 使用 Canny 检测边缘
    Canny( g_grayImage, g_cannyMat_output, g_nThresh, g_nThresh*2, 3 );

    // 找到轮廓
    findContours( g_cannyMat_output, g_vContours, g_vHierarchy,
RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0) );

    // 计算矩
    vector<Moments> mu(g_vContours.size());
    for(unsigned int i = 0; i < g_vContours.size(); i++)
    { mu[i] = moments( g_vContours[i], false ); }

    // 计算中心矩
    vector<Point2f> mc( g_vContours.size());
    for( unsigned int i = 0; i < g_vContours.size(); i++)
    { mc[i] = Point2f( static_cast<float>(mu[i].m10/mu[i].m00),
static_cast<float>(mu[i].m01/mu[i].m00) ); }

    // 绘制轮廓
    Mat drawing = Mat::zeros( g_cannyMat_output.size(), CV_8UC3 );
    for( unsigned int i = 0; i < g_vContours.size(); i++)
    {
        Scalar color = Scalar( g_rng.uniform(0, 255),
g_rng.uniform(0,255), g_rng.uniform(0,255) );//随机生成颜色值
        drawContours( drawing, g_vContours, i, color, 2, 8, g_vHierarchy,
0, Point() );//绘制外层和内层轮廓
        circle( drawing, mc[i], 4, color, -1, 8, 0 );//绘制圆
    }
}
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
// 显示到窗口中
namedWindow( WINDOW_NAME2, WINDOW_AUTOSIZE );
imshow( WINDOW_NAME2, drawing );

// 通过 m00 计算轮廓面积并且和 OpenCV 函数比较
printf("\t 输出内容: 面积和轮廓长度\n");
for(unsigned int i = 0; i< g_vContours.size(); i++)
{
    printf(" >通过 m00 计算出轮廓[%d] 的面积: (M_00) = %.2f \n OpenCV 函
数计算出的面积= %.2f , 长度: %.2f \n\n", i, mu[i].m00,
contourArea(g_vContours[i]), arcLength( g_vContours[i], true ) );
    Scalar color = Scalar( g_rng.uniform(0, 255),
g_rng.uniform(0,255), g_rng.uniform(0,255) );
    drawContours( drawing, g_vContours, i, color, 2, 8, g_vHierarchy,
0, Point() );
    circle( drawing, mc[i], 4, color, -1, 8, 0 );
}
}
```

运行程序，便可检测和计算出各个区域的轮廓面积和长度，在窗口中输出。如图 8.25~8.29 所示。

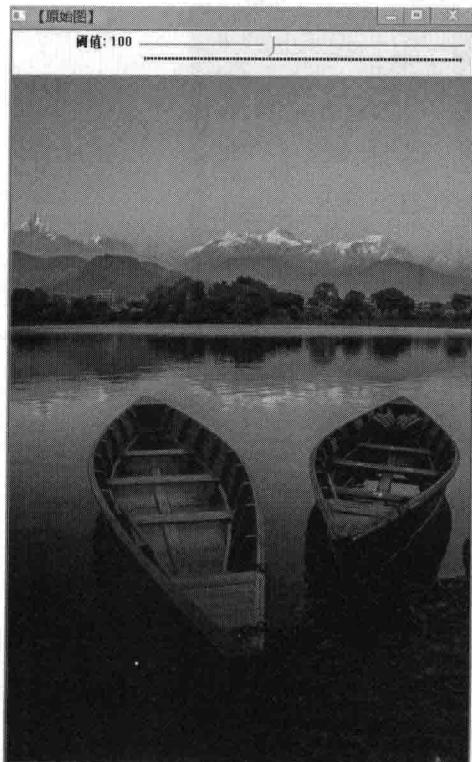


图 8.25 原始图

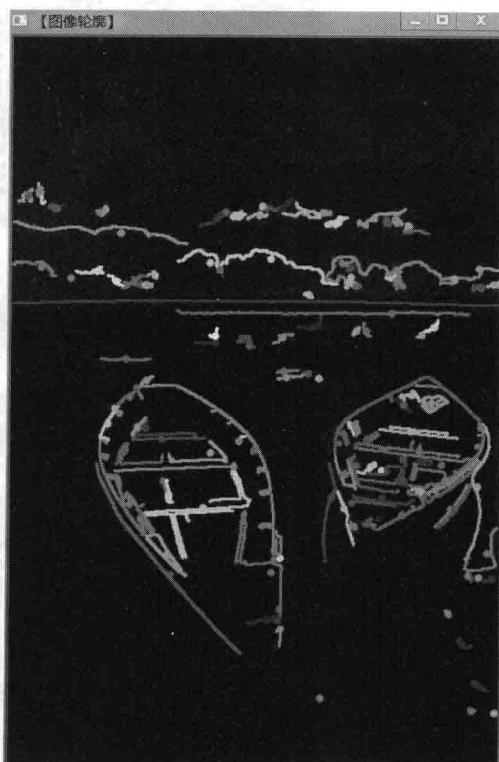
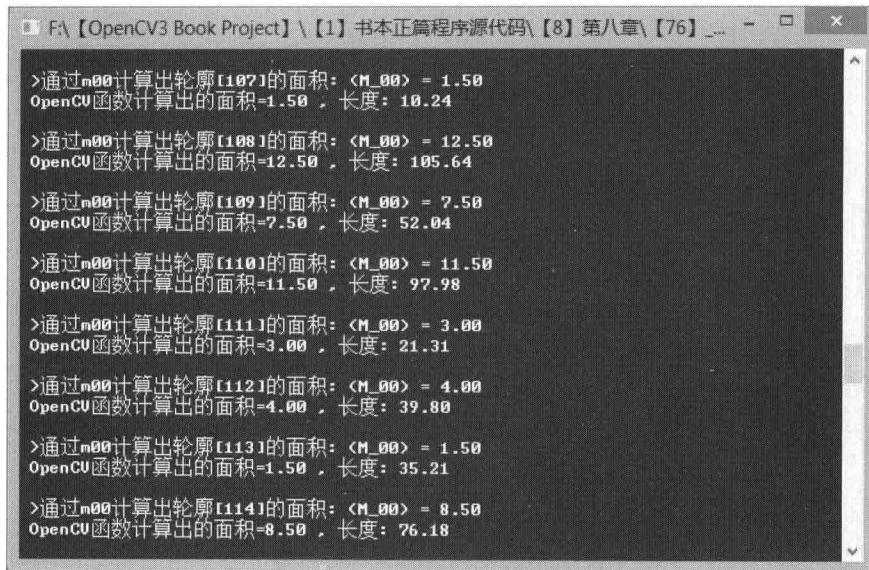


图 8.26 阈值为 100 时的图像轮廓

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>



```
>通过m00计算出轮廓[107]的面积: <M_00> = 1.50  
OpenCV函数计算出的面积=1.50 , 长度: 10.24  
  
>通过m00计算出轮廓[108]的面积: <M_00> = 12.50  
OpenCV函数计算出的面积=12.50 , 长度: 105.64  
  
>通过m00计算出轮廓[109]的面积: <M_00> = 7.50  
OpenCV函数计算出的面积=7.50 , 长度: 52.04  
  
>通过m00计算出轮廓[110]的面积: <M_00> = 11.50  
OpenCV函数计算出的面积=11.50 , 长度: 97.98  
  
>通过m00计算出轮廓[111]的面积: <M_00> = 3.00  
OpenCV函数计算出的面积=3.00 , 长度: 21.31  
  
>通过m00计算出轮廓[112]的面积: <M_00> = 4.00  
OpenCV函数计算出的面积=4.00 , 长度: 39.88  
  
>通过m00计算出轮廓[113]的面积: <M_00> = 1.50  
OpenCV函数计算出的面积=1.50 , 长度: 35.21  
  
>通过m00计算出轮廓[114]的面积: <M_00> = 8.50  
OpenCV函数计算出的面积=8.50 , 长度: 76.18
```

图 8.27 阈值为 100 时的输出窗口

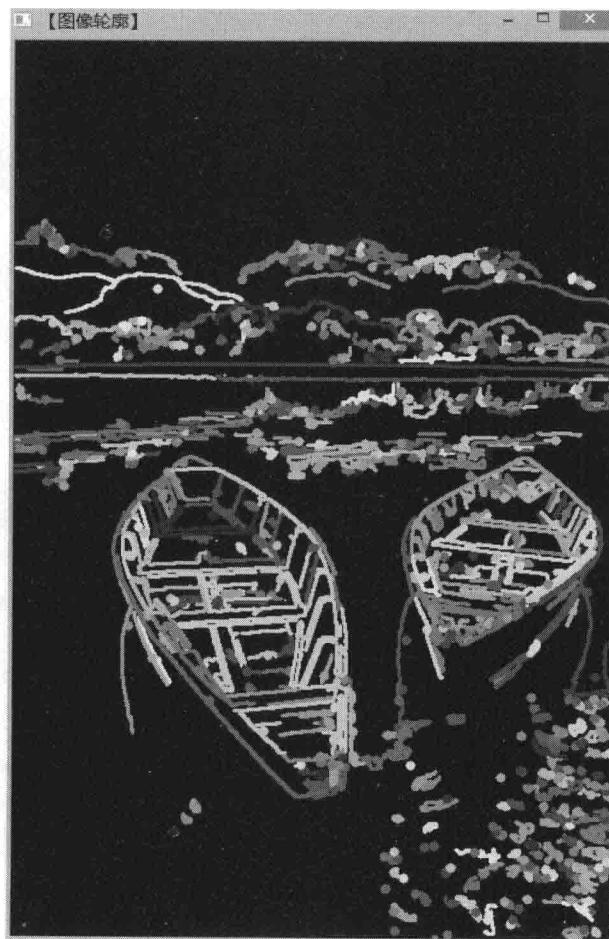


图 8.28 阈值为 40 时的图像轮廓

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

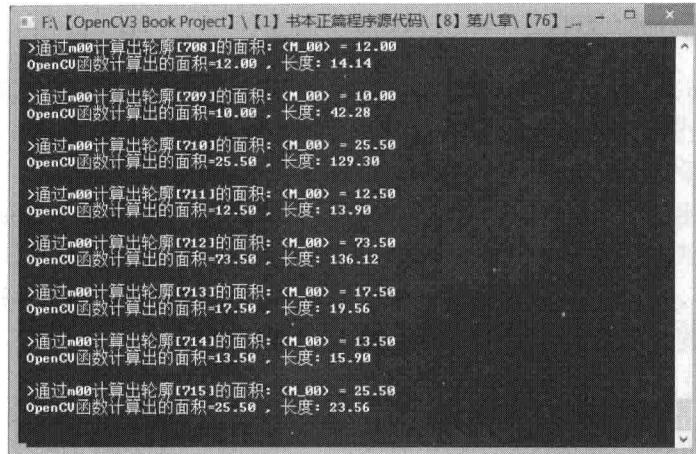


图 8.29 阈值为 40 时的输出窗口

## 8.5 分水岭算法

在许多实际运用中，我们需要分割图像，但无法从背景图像中获得有用信息。分水岭算法（watershed algorithm）在这方面往往是非常有效的。此算法可以将图像中的边缘转化成“山脉”，将均匀区域转化为“山谷”，这样有助于分割目标。

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。

分水岭的计算过程是一个迭代标注过程。分水岭比较经典的计算方法是由 L. Vincent 提出的。在该算法中，分水岭计算分两个步骤：一个是排序过程，一个是淹没过程。首先对每个像素的灰度级进行从低到高的排序，然后在从低到高实现淹没的过程中，对每一个局部极小值在  $h$  阶高度的影响域采用先进先出（FIFO）结构进行判断及标注。分水岭变换得到的是输入图像的集水盆图像，集水盆之间的边界点，即为分水岭。显然，分水岭表示的是输入图像的极大值点。

也就是说，分水岭算法首先计算灰度图像的梯度；这对图像中的“山谷”或没有纹理的“盆地”（亮度值低的点）的形成是很有效的，也对“山头”或图像中有主导线段的“山脉”（山脊对应的边缘）的形成有效。然后开始从用户指定点（或者算法得到点）开始持续“灌注”盆地直到这些区域连成一片。基于这样产生的标记就可以把区域合并到 0 一起，合并后的区域又通聚集的方式进行分割，好像图像被“填充”起来一样。

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

### 8.5.1 实现分水岭算法：watershed()函数

函数 watershed 实现的分水岭算法是基于标记的分割算法中的一种。在把图像传给函数之前，我们需要大致勾画标记出图像中的期望进行分割的区域，它们被标记为正指数。所以，每一个区域都会被标记为像素值 1、2、3 等，表示成为一个或者多个连接组件。这些标记的值可以使用 findContours() 函数和 drawContours() 函数由二进制的掩码检索出来。不难理解，这些标记就是即将绘制出来的分割区域的“种子”，而没有标记清楚的区域，被置为 0。在函数输出中，每一个标记中的像素被设置为“种子”的值，而区域间的值被设置为 -1。

```
C++: void watershed(InputArray image, InputOutputArray markers)
```

第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可，且需为 8 位三通道的彩色图像。

第二个参数，InputOutputArray 类型的 markers，函数调用后的运算结果存在这里，输入/输出 32 位单通道图像的标记结果。即这个参数用于存放函数调用后的输出结果，需和源图片有一样的尺寸和类型。

### 8.5.2 综合示例程序：分水岭算法

```
-----【头文件、命名空间包含部分】-----
//    描述：包含程序所依赖的头文件和命名空间
//-----  

#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
using namespace cv;
using namespace std;  

-----【宏定义部分】-----
//    描述：定义一些辅助宏
//-----  

#define WINDOW_NAME "【程序窗口 1】"      //为窗口标题定义的宏  

-----【全局函变量声明部分】-----
//    描述：全局变量的声明
//-----  

Mat g_maskImage, g_srcImage;
Point prevPt(-1, -1);  

-----【全局函数声明部分】-----
//    描述：全局函数的声明
//-----  

static void ShowHelpText();
static void on_Mouse( int event, int x, int y, int flags, void* );  

-----【main()函数】-----
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----
int main( int argc, char** argv )
{
    //【1】载入原图并显示，初始化掩膜和灰度图
    g_srcImage = imread("1.jpg", 1);
    imshow( WINDOW_NAME, g_srcImage );
    Mat srcImage,grayImage;
    g_srcImage.copyTo(srcImage);
    cvtColor(g_srcImage, g_maskImage, COLOR_BGR2GRAY);
    cvtColor(g_maskImage, grayImage, COLOR_GRAY2BGR);
    g_maskImage = Scalar::all(0);

    //【2】设置鼠标回调函数
    setMouseCallback( WINDOW_NAME, on_Mouse, 0 );

    //【3】轮询按键，进行处理
    while(1)
    {
        //获取键值
        int c = waitKey(0);

        //若按键键值为 ESC 时，退出
        if( (char)c == 27 )
            break;

        //按键键值为 2 时，恢复源图
        if( (char)c == '2' )
        {
            g_maskImage = Scalar::all(0);
            srcImage.copyTo(g_srcImage);
            imshow( "image", g_srcImage );
        }

        //若检测到按键值为 1 或者空格，则进行处理
        if( (char)c == '1' || (char)c == ' ' )
        {
            //定义一些参数
            int i, j, compCount = 0;
            vector<vector<Point>> contours;
            vector<Vec4i> hierarchy;

            //寻找轮廓
            findContours(g_maskImage, contours, hierarchy, CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE);

            //轮廓为空时的处理
            if( contours.empty() )
                continue;

            //复制掩膜
            Mat maskImage(g_maskImage.size(), CV_32S);
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
maskImage = Scalar::all(0);

//循环绘制出轮廓
for( int index = 0; index >= 0; index = hierarchy[index][0],
compCount++ )
    drawContours(maskImage, contours, index,
Scalar::all(compCount+1), -1, 8, hierarchy, INT_MAX);

//compCount 为零时的处理
if( compCount == 0 )
    continue;

//生成随机颜色
vector<Vec3b> colorTab;
for( i = 0; i < compCount; i++ )
{
    int b = theRNG().uniform(0, 255);
    int g = theRNG().uniform(0, 255);
    int r = theRNG().uniform(0, 255);

    colorTab.push_back(Vec3b((uchar)b, (uchar)g, (uchar)r));
}

//计算处理时间并输出到窗口中
double dTime = (double)getTickCount();
watershed( srcImage, maskImage );
dTime = (double)getTickCount() - dTime;
printf( "\t 处理时间 = %gms\n",
dTime*1000./getTickFrequency() );

//双层循环，将分水岭图像遍历存入 watershedImage 中
Mat watershedImage(maskImage.size(), CV_8UC3);
for( i = 0; i < maskImage.rows; i++ )
    for( j = 0; j < maskImage.cols; j++ )
    {
        int index = maskImage.at<int>(i,j);
        if( index == -1 )
            watershedImage.at<Vec3b>(i,j) =
Vec3b(255,255,255);
        else if( index <= 0 || index > compCount )
            watershedImage.at<Vec3b>(i,j) = Vec3b(0,0,0);
        else
            watershedImage.at<Vec3b>(i,j) = colorTab[index - 1];
    }

//混合灰度图和分水岭效果图并显示最终的窗口
watershedImage = watershedImage*0.5 + grayImage*0.5;
imshow( "watershed transform", watershedImage );
}
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```

    return 0;
}

//-----【onMouse() 函数】-----
//      描述：鼠标消息回调函数
//-----
static void on_Mouse( int event, int x, int y, int flags, void* )
{
    //处理鼠标不在窗口中的情况
    if( x < 0 || x >= g_srcImage.cols || y < 0 || y >= g_srcImage.rows )
        return;

    //处理鼠标左键相关消息
    if( event == EVENT_LBUTTONUP || !(flags & EVENT_FLAG_LBUTTON) )
        prevPt = Point(-1,-1);
    else if( event == EVENT_LBUTTONDOWN )
        prevPt = Point(x,y);

    //鼠标左键按下并移动，绘制出白色线条
    else if( event == EVENT_MOUSEMOVE && (flags & EVENT_FLAG_LBUTTON) )
    {
        Point pt(x, y);
        if( prevPt.x < 0 )
            prevPt = pt;
        line( g_maskImage, prevPt, pt, Scalar::all(255), 5, 8, 0 );
        line( g_srcImage, prevPt, pt, Scalar::all(255), 5, 8, 0 );
        prevPt = pt;
        imshow(WINDOW_NAME, g_srcImage);
    }
}
}

```

程序运行截图如图 8.30~8.33 所示。

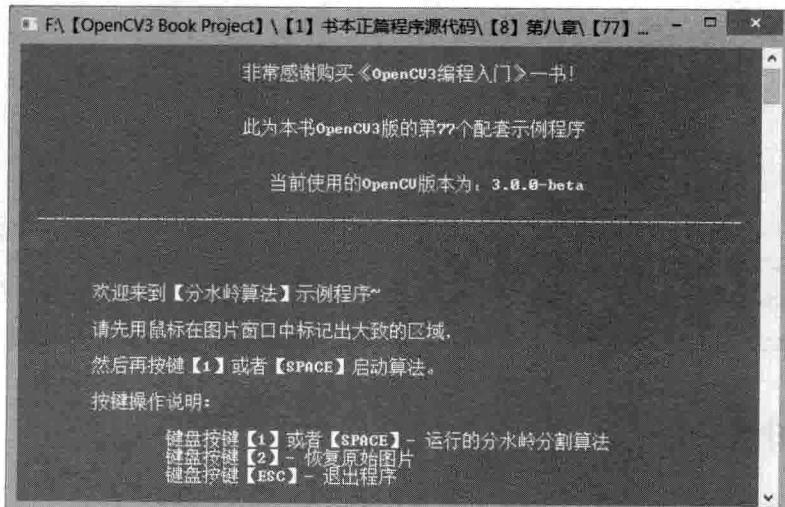


图 8.30 说明窗口



图 8.31 原始图窗口



图 8.32 用鼠标勾勒出区域后的图



图 8.33 分水岭算法效果图

## 8.6 图像修补

在实际应用中，我们的图像常常会被噪声腐蚀，这些噪声或者是镜头上的灰尘或水滴，或者是旧照片的划痕，或者由于图像的部分本身已经损坏。而“图像修复”(Inpainting)，就是妙手回春，解决这些问题的良方。图像修复技术简单来

说，就是利用那些已经被破坏区域的边缘，即边缘的颜色和结构，繁殖和混合到损坏的图像中，以达到图像修补的目的。图 8.34~8.36 就是示例程序截图，演示将图像中的字迹移除的效果。



由于黑白印刷的缘故，本节书本上的图实际显示效果可能会受影响。请大家找到本节的示例程序（程序序号为 78）自己编译运行，进行图像修补技术的体验。



图 8.34 原始图



图 8.35 含字迹的图



图 8.36 经过修补后的图

如果被破坏的区域不是太大，并且在被破坏区域边缘包含足够多的纹理和颜色，那么图像修补技术可以很好地恢复图像。当然，当图像损坏区域过大时，我们“妙手回春”的能力也是有限的。如图 8.37、8.38 所示。

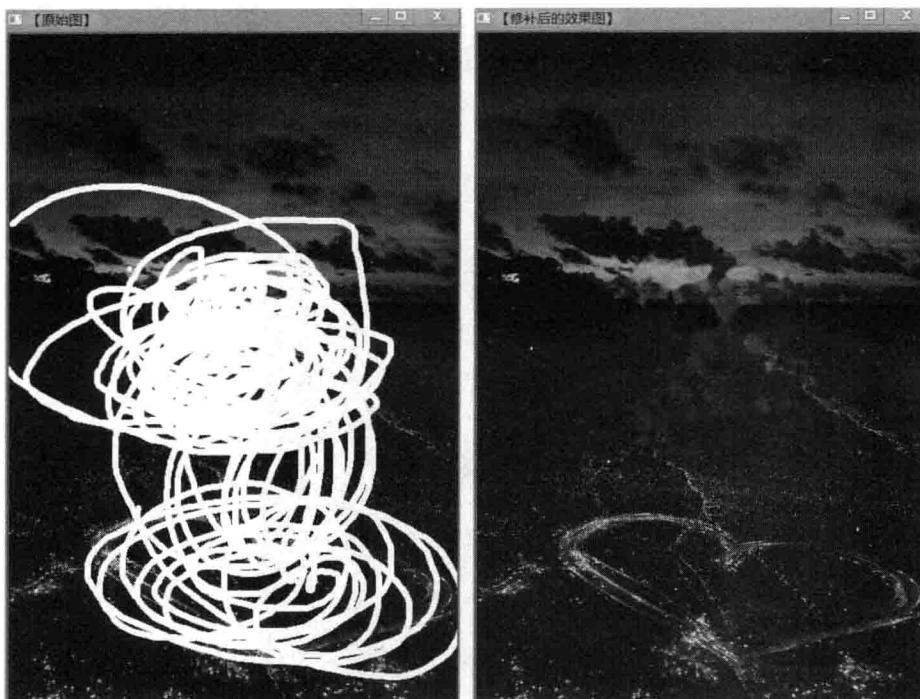


图 8.37 受过过大破坏后的图

图 8.38 经过修补后的图

### 8.6.1 实现图像修补：inpaint()函数

在新版 OpenCV 中，图像修补技术由 `inpaint` 函数实现，它可以用来从扫描的照片中清除灰尘和划痕，或者从静态图像或视频中去除不需要的物体。其原型声明如下。

```
C++: void inpaint(InputArray src, InputArray inpaintMask, OutputArray dst, double inpaintRadius, int flags)
```

- 第一个参数，`InputArray` 类型的 `src`，输入图像，即源图像，填 `Mat` 类的对象即可，且需为 8 位单通道或者三通道图像。
- 第二个参数，`InputArray` 类型的 `inpaintMask`，修复掩膜，为 8 位的单通道图像。其中的非零像素表示需要修补的区域。
- 第三个参数，`OutputArray` 类型的 `dst`，函数调用后的运算结果存在这里，和源图片有一样的尺寸和类型。
- 第四个参数，`double` 类型的 `inpaintRadius`，需要修补的每个点的圆形邻域，为修复算法的参考半径。
- 第五个参数，`int` 类型的 `flags`，修补方法的标识符，可以是表 8.4 所示两者之一。

表 8.4 图像修补方法的标识符

标识符	说明
INPAINT_NS	基于 Navier-Stokes 方程的方法
INPAINT_TELEA	Alexandru Telea 方法



OpenCV2 中 INPAINT\_NS 和 INPAINT\_TELEA 标识符可以分别写作 CV\_INPAINT\_NS 和 CV\_INPAINT\_TELEA

### 8.6.2 综合示例程序：图像修补

函数和概念讲解完毕，下面我们依然是学习一个以本节所讲内容为核心的示例程序，将本节所学内容付诸实践，融会贯通。此示例程序会先让我们在图像中用鼠标绘制出白色的线条破坏图像，然后按下键盘按键【1】或【SPACE】进行图像修补操作。且如果对自己的绘制不够满意，可以按下键盘按键【2】恢复原始图像。操作说明如图 8.39 所示。

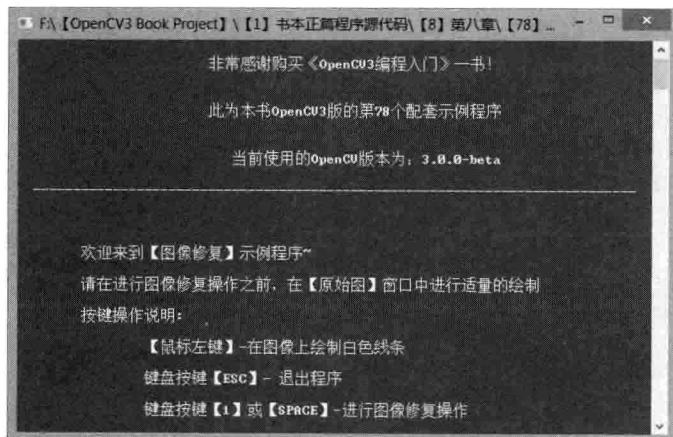


图 8.39 程序操作说明

```

-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----  

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/photo/photo.hpp"
#include <iostream>
using namespace cv;
using namespace std;

-----【宏定义部分】-----
//      描述：定义一些辅助宏
//-----  

#define WINDOW_NAME1 "【原始图】"          //为窗口标题定义的宏
#define WINDOW_NAME2 "【修补后的效果图】"    //为窗口标题定义的宏

```

```
-----【全局变量声明部分】-----
//      描述：全局变量声明
-----

Mat srcImage1, inpaintMask;
Point previousPoint(-1,-1); //原来的点坐标

-----【On_Mouse() 函数】-----
//      描述：响应鼠标消息的回调函数
-----

static void On_Mouse( int event, int x, int y, int flags, void* )
{
    //鼠标左键弹起消息
    if( event == EVENT_LBUTTONUP || !(flags & EVENT_FLAG_LBUTTON) )
        previousPoint = Point(-1,-1);
    //鼠标左键按下消息
    else if( event == EVENT_LBUTTONDOWN )
        previousPoint = Point(x,y);
    //鼠标按下并移动，进行绘制
    else if( event == EVENT_MOUSEMOVE && (flags & EVENT_FLAG_LBUTTON) )
    {
        Point pt(x,y);
        if( previousPoint.x < 0 )
            previousPoint = pt;
        //绘制白色线条
        line( inpaintMask, previousPoint, pt, Scalar::all(255), 5, 8,
0 );
        line( srcImage1, previousPoint, pt, Scalar::all(255), 5, 8, 0 );
        previousPoint = pt;
        imshow(WINDOW_NAME1, srcImage1);
    }
}

-----【main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
-----

int main( int argc, char** argv )
{
    //载入原始图并进行掩膜的初始化
    Mat srcImage = imread("1.jpg", -1);
    if(!srcImage.data) { printf("读取图片错误，请确定目录下是否有 imread
函数指定图片存在~! \n"); return false; }
    srcImage1 = srcImage.clone();
    inpaintMask = Mat::zeros(srcImage1.size(), CV_8U);

    //显示原始图
    imshow(WINDOW_NAME1, srcImage1);

    //设置鼠标回调消息
    setMouseCallback( WINDOW_NAME1, On_Mouse, 0 );
}
```

```

//轮询按键，根据不同的按键进行处理
while (1)
{
    //获取按键键值
    char c = (char)waitKey();

    //键值为 ESC，程序退出
    if( c == 27 )
        break;

    //键值为 2，恢复成原始图像
    if( c == '2' )
    {
        inpaintMask = Scalar::all(0);
        srcImage.copyTo(srcImage1);
        imshow(WINDOW_NAME1, srcImage1);
    }

    //键值为 1 或者空格，进行图像修补操作
    if( c == '1' || c == ' ' )
    {
        Mat inpaintedImage;
        inpaint(srcImage1, inpaintMask, inpaintedImage, 3,
INPAINT_TELEA);
        imshow(WINDOW_NAME2, inpaintedImage);
    }
}

return 0;
}

```

程序的运行截图在概念讲解部分已经贴出过，在这里就不再贴出。请参考图 8.34~图 8.39。

## 8.7 本章小结

本章中，我们先学习了查找轮廓并绘制轮廓，然后学习了如何寻找到物体的凸包，接着是使用多边形来包围轮廓，以及计算一个图像的矩。在本章后面几节，还学习了分水岭算法和图像修补操作的实现方法。

### 本章核心函数清单

函数名称	说明	对应讲解章节
findContours	在二值图像中寻找轮廓	8.1.1
drawContours	在图像中绘制外部或内部轮廓	8.1.2
convexHull	寻找图像点集中的凸包	8.2.2

续表

函数名称	说明	对应讲解章节
BoundingRect	计算并返回指定点集最外面（up-right）的矩形边界	8.3.1
minAreaRect	寻找可旋转的最小面积的包围矩形	8.3.2
minEnclosingCircle	利用一种迭代算法，对给定的 2D 点集，寻找面积最小的可包围他们的圆形	8.3.3
fitEllipse	用椭圆拟合二维点集	8.3.4
approxPolyDP	用指定精度逼近多边形曲线	8.3.5
moments	计算多边形和光栅形状的最高达三阶的所有矩	8.4.1
contourArea	计算整个轮廓或部分轮廓的面积	8.4.2
arcLength	计算封闭轮廓的周长或曲线的长度	8.4.3
watershed	实现分水岭算法	8.5.1
inpaint	进行图像修补，从扫描的照片中清除灰尘和划痕，或者从静态图像或视频中去除不需要的物体	8.6.1

### 本章示例程序清单

示例程序序号	程序说明	对应章节
69	轮廓查找	8.1.3
70	查找并绘制轮廓	8.1.4
71	凸包检测基础	8.2.3
72	寻找和绘制物体的凸包	8.2.4
73	创建包围轮廓的矩形边界	8.3.6
74	创建包围轮廓的圆形边界	8.3.7
75	使用多边形包围轮廓	8.3.8
76	图像轮廓矩	8.4.4
77	分水岭算法的使用	8.5.2
78	实现图像修补	8.6.2

# 第 9 章

## 直方图与匹配

### 导读

---

在进行物体图像和视频信息分析的过程中，我们常常会习惯于将眼中看到的物体用直方图（histogram）表示出来，得到比较直观的数据官感展示。直方图可以用来描述各种不同的参数和事物，如物体的色彩分布、物体边缘梯度模板，以及表示目标位置的当前假设的概率分布。

### 本章你将学到：

---

- 什么是直方图
- 直方图的计算与绘制
- 如何进行直方图的对比
- 反向投影技术
- 模板匹配技术

## 9.1 图像直方图概述

直方图广泛运用于很多计算机视觉运用当中，通过标记帧与帧之间显著的边缘和颜色的统计变化，来检测视频中场景的变化。在每个兴趣点设置一个有相近特征的直方图所构成“标签”，用以确定图像中的兴趣点。边缘、色彩、角度等直方图构成了可以被传递给目标识别分类器的一个通用特征类型。色彩和边缘的直方图序列还可以用来识别网络视频是否被复制。如图 9.1 所示。

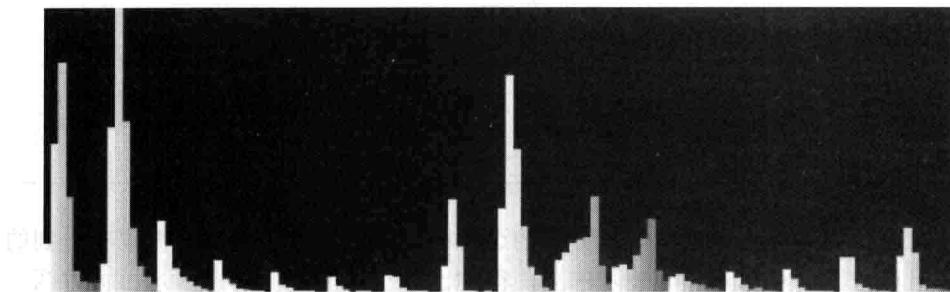


图 9.1 某图像的 H-S 颜色直方图

其实，简单点说，直方图就是对数据进行统计的一种方法，并且将统计值组织到一系列事先定义好的 bin 当中。其中，bin 为直方图中经常用到的一个概念，可翻译为“直条”或“组距”，其数值是从数据中计算出的特征统计量，这些数据可以是诸如梯度、方向、色彩或任何其他特征。且无论如何，直方图获得的是数据分布的统计图。通常直方图的维数要低于原始数据。总而言之，直方图是计算机视觉中最经典的工具之一。

在统计学中，直方图（Histogram）是一种对数据分布情况的图形表示，是一种二维统计图表，它的两个坐标分别是统计样本和该样本对应的某个属性的度量。

我们在图像变换的那一章中讲过直方图的均衡化，它是通过拉伸像素强度分布范围来增强图像对比度的一种方法。大家在自己的心目中应该已经对直方图有一定的理解和认知。下面就来看一看对图像直方图比较书面化的解释。

图像直方图（Image Histogram）是用以表示数字图像中亮度分布的直方图，标绘了图像中每个亮度值的像素数。可以借助观察该直方图了解需要如何调整亮度分布。这种直方图中，横坐标的左侧为纯黑、较暗的区域，而右侧为较亮、纯白的区域。因此，一张较暗图片的图像直方图中的数据多集中于左侧和中间部分，而整体明亮、只有少量阴影的图像则相反。计算机视觉领域常借助图像直方图来实现图像的二值化。

直方图的意义如下。

- 直方图是图像中像素强度分布的图形表达方式。
- 它统计了每一个强度值所具有的像素个数。

上面已经讲到，直方图是对数据的统计集合，并将统计结果分布于一系列预

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

定义的 bins 中。这里的数据不仅仅指的是灰度值，且统计数据可能是任何能有效描述图像的特征。下面看一个例子，假设有一个矩阵包含一张图像的信息（灰度值 0-255），让我们按照某种方式来统计这些数字。既然已知数字的范围包含 256 个值，于是可以将这个范围分割成子区域（也就是上面讲到 bins），如：

$$[0,255] = [0,15] \cup [16,31] \cup \dots \cup [240,255]$$

$$\text{range} = \text{bin}_1 \cup \text{bin}_2 \cup \dots \cup \text{bin}_{n=15}$$

然后再统计每一个  $\text{bin}_i$  的像素数目。采用这一方法来统计上面的数字矩阵，可以得到图 9.2（其中 x 轴表示 bin，y 轴表示各个 bin 中的像素个数）。

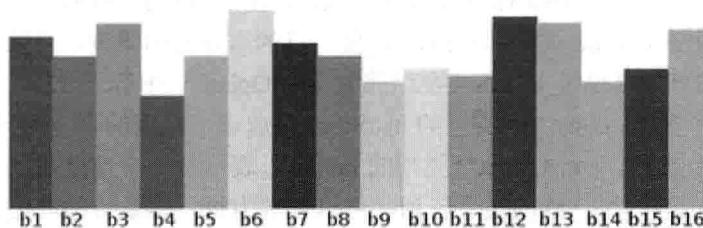


图 9.2 图像数字矩阵的直方图

以上就是一个说明直方图的用途的简单示例。其实，直方图并不局限于统计颜色灰度，而是可以统计任何图像特征，如梯度、方向等。

让我们具体讲讲直方图的一些术语和细节。

- dims：需要统计的特征的数目。在上例中，`dims = 1` 因为我们仅仅统计了灰度值(灰度图像)。
- bins：每个特征空间子区段的数目，可翻译为“直条”或“组距”。在上例中，`bins = 16`。
- range：每个特征空间的取值范围。在上例中，`range = [0,255]`。

## 9.2 直方图的计算与绘制

直方图的计算在 OpenCV 中可以使用 `calcHist()` 函数，而计算完成之后，可以采用 OpenCV 中的绘图函数，如绘制矩形的 `rectangle()` 函数，绘制线段的 `line()` 来完成。

### 9.2.1 计算直方图：`calcHist()` 函数

在 OpenCV 中，`calcHist()` 函数用于计算一个或者多个阵列的直方图。原型如下。

```
C++: void calcHist(const Mat* images, int nimages, const int* channels,
InputArray mask, OutputArray hist, int dims, const int* histSize, const
float** ranges, bool uniform=true, bool accumulate=false )
```

- 第一个参数，`const Mat*`类型的 `images`，输入的数组(或数组集)，它们需为

[Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com](http://www.simpopdf.com)

相同的深度 (CV\_8U 或 CV\_32F) 和相同的尺寸。

- 第二个参数, int 类型的 nimages, 输入数组的个数, 也就是第一个参数中存放了多少张“图像”, 有几个原数组。
- 第三个参数, const int\*类型的 channels, 需要统计的通道 (dim) 索引。第一个数组通道从 0 到 images[0].channels()-1, 而第二个数组通道从 images[0].channels() 计算到 images[0].channels() + images[1].channels()-1。
- 第四个参数, InputArray 类型的 mask, 可选的操作掩码。如果此掩码不为空, 那么它必须为 8 位, 并且与 images[i] 有同样大小的尺寸。这里的非零掩码元素用于标记出统计直方图的数组元素数据。
- 第五个参数, OutputArray 类型的 hist, 输出的目标直方图, 一个二维数组。
- 第六个参数, int 类型的 dims, 需要计算的直方图的维度, 必须是正数, 且不大于 CV\_MAX\_DIMS (在当前版本的 OpenCV 中等于 32)。
- 第七个参数, const int\*类型的 histSize, 存放每个维度的直方图尺寸的数组。
- 第八个参数, const float\*\*类型的 ranges, 表示每一个维度数组 (第六个参数 dims) 的每一维的边界阵列, 可以理解为每一维数值的取值范围。
- 第九个参数, bool 类型的 uniform, 指示直方图是否均匀的标识符, 有默认值 true。
- 第十个参数, bool 类型的 accumulate, 累计标识符, 有默认值 false。若其为 true, 直方图在配置阶段不会被清零。此功能主要是允许从多个阵列中计算单个直方图, 或者用于在特定的时间更新直方图。

### 9.2.2 找寻最值: minMaxLoc()函数

minMaxLoc()函数的作用是在数组中找到全局最小值和最大值。它有两个版本的原型, 在此介绍常用的那一个版本。

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

- 第一个参数, InputArray 类型的 src, 输入的单通道阵列。
- 第二个参数, double\*类型的 minVal, 返回最小值的指针。若无须返回, 此值置为 NULL。
- 第三个参数, double\*类型的 maxVal, 返回的最大值的指针。若无须返回, 此值置为 NULL。
- 第四个参数, Point\*类型的 minLoc, 返回最小位置的指针 (二维情况下)。若无须返回, 此值置为 NULL。
- 第五个参数, Point\*类型的 maxLoc, 返回最大位置的指针 (二维情况下)。若无须返回, 此值置为 NULL。
- 第六个参数, InputArray 类型的 mask, 用于选择子阵列的可选掩膜。

### 9.2.3 示例程序: 绘制 H—S 直方图

下面的示例说明如何计算彩色图像的色调, 饱和度二维直方图。

**SimpPDF Merge and Split Unregistered Version - http://www.simpopdf.com**

注意：色调（Hue），饱和度（Saturation）。所以“H-S 直方图”就是“色调—饱和度直方图”。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;

-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
-----
int main()
{
    //【1】载入源图，转化为 HSV 颜色模型
    Mat srcImage, hsvImage;
    srcImage=imread("1.jpg");
    cvtColor(srcImage,hsvImage, COLOR_BGR2HSV);

    //【2】参数准备
    //将色调量化为 30 个等级，将饱和度量化为 32 个等级
    int hueBinNum = 30;//色调的直方图直条数量
    int saturationBinNum = 32;//饱和度的直方图直条数量
    int histSize[] = {hueBinNum, saturationBinNum};
    // 定义色调的变化范围为 0 到 179
    float hueRanges[] = { 0, 180 };
    //定义饱和度的变化范围为 0 (黑、白、灰) 到 255 (纯光谱颜色)
    float saturationRanges[] = { 0, 256 };
    const float* ranges[] = { hueRanges, saturationRanges };
    MatND dstHist;
    //参数准备，calcHist 函数中将计算第 0 通道和第 1 通道的直方图
    int channels[] = {0, 1};

    //【3】正式调用 calcHist，进行直方图计算
    calcHist( &hsvImage,//输入的数组
               1, //数组个数为 1
               channels,//通道索引
               Mat(), //不使用掩膜
               dstHist, //输出的目标直方图
               2, //需要计算的直方图的维度为 2
               histSize, //存放每个维度的直方图尺寸的数组
               ranges,//每一维数值的取值范围数组
               true, // 指示直方图是否均匀的标识符，true 表示均匀的直方图
               false );//累计标识符，false 表示直方图在配置阶段会被清零

    //【4】为绘制直方图准备参数
    double maxValue=0;//最大值
    minMaxLoc(dstHist, 0, &maxValue, 0, 0); //查找数组和子数组的全局最小值
    和最大值存入 maxValue 中
    int scale = 10;
```

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

    Mat histImg = Mat::zeros(saturationBinNum*scale, hueBinNum*10,
CV_8UC3);

    //【5】双层循环，进行直方图绘制
    for( int hue = 0; hue < hueBinNum; hue++ )
        for( int saturation = 0; saturation < saturationBinNum;
saturation++ )
    {
        float binValue = dstHist.at<float>(hue, saturation); //直方图直
条的值
        int intensity = cvRound(binValue*255/maxValue); //强度

        //正式进行绘制
        rectangle( histImg, Point(hue*scale, saturation*scale),
Point( (hue+1)*scale - 1, (saturation+1)*scale - 1),
Scalar::all(intensity), FILLED );
    }

    //【6】显示效果图
    imshow( "素材图", srcImage );
    imshow( "H-S 直方图", histImg );

    waitKey();
}

```

程序中新出现的 MatND 类是用于存储直方图的一种数据结构，其用法简单，常常在直方图相关 OpenCV 程序中出现。而+-\*\*程序运行截图如图 9.4、9.5 所示。



图 9.3 素材图



图 9.4 图像的 H-S 二维直方图

### 9.2.4 示例程序：计算并绘制图像一维直方图

上文中已经讲解了 calcHist() 函数的用法，并绘制出了图像的 H-S 二维直方图。而本节我们会通过一个示例，来学习图像一维直方图的计算和绘制过程。

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
//-----  
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;  
  
-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----  
int main()
{
    //【1】载入原图并显示
    Mat srcImage = imread("1.jpg", 0);
    imshow("原图", srcImage);
    if(!srcImage.data) {cout << "fail to load image" << endl; return 0;}  
  
    //【2】定义变量
    MatND dstHist;          // 在 cv 中用 CvHistogram *hist = cvCreateHist
    int dims = 1;
    float hranges[] = {0, 255};
    const float *ranges[] = {hranges}; // 这里需要为 const 类型
    int size = 256;
    int channels = 0;  
  
    //【3】计算图像的直方图
    calcHist(&srcImage, 1, &channels, Mat(), dstHist, dims, &size,
ranges); // cv 中是 cvCalcHist
    int scale = 1;  
  
    Mat dstImage(size * scale, size, CV_8U, Scalar(0));
    //【4】获取最大值和最小值
    double minValue = 0;
    double maxValue = 0;
    minMaxLoc(dstHist,&minValue, &maxValue, 0, 0); // 在 cv 中用的是
cvGetMinMaxHistValue  
  
    //【5】绘制出直方图
    int hpt = saturate_cast<int>(0.9 * size);
    for(int i = 0; i < 256; i++)
    {
        float binValue = dstHist.at<float>(i);           // 注意 hist 中
是 float 类型 而在 OpenCV1.0 版中用 cvQueryHistValue_1D
        int realValue = saturate_cast<int>(binValue * hpt/maxValue);
        rectangle(dstImage,Point(i*scale, size - 1), Point((i+1)*scale
- 1, size - realValue), Scalar(255));
    }
    imshow("一维直方图", dstImage);
    waitKey(0);
```

```
    return 0;  
}
```

运行此详细注释的代码，可以得到如图 9.5、9.6 所示的运行结果。



图 9.5 素材图

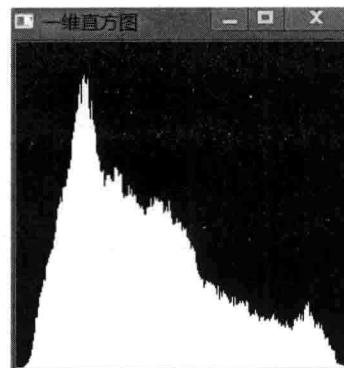


图 9.6 素材图的一维直方图

### 9.2.5 示例程序：绘制 RGB 三色直方图

上文我们讲解的是单个分量的一维直方图的绘制，接下来看看如何分别绘制图像的 RGB 三色直方图。详细注释的代码如下。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所使用的头文件和命名空间  
-----  
#include <opencv2/opencv.hpp>  
#include <opencv2/imgproc/imgproc.hpp>  
using namespace cv;  
  
-----【main()函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行  
-----  
int main()  
{  
    //【1】载入素材图并显示  
    Mat srcImage;  
    srcImage=imread("1.jpg");
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
imshow( "素材图", srcImage );

//【2】参数准备
int bins = 256;
int hist_size[] = {bins};
float range[] = { 0, 256 };
const float* ranges[] = { range };
MatND redHist,grayHist,blueHist;
int channels_r[] = {0};

//【3】进行直方图的计算（红色分量部分）
calcHist( &srcImage, 1, channels_r, Mat(), //不使用掩膜
           redHist, 1, hist_size, ranges,
           true, false );

//【4】进行直方图的计算（绿色分量部分）
int channels_g[] = {1};
calcHist( &srcImage, 1, channels_g, Mat(), // do not use mask
           grayHist, 1, hist_size, ranges,
           true, // the histogram is uniform
           false );

//【5】进行直方图的计算（蓝色分量部分）
int channels_b[] = {2};
calcHist( &srcImage, 1, channels_b, Mat(), // do not use mask
           blueHist, 1, hist_size, ranges,
           true, // the histogram is uniform
           false );

//-----绘制出三色直方图-----
//参数准备
double maxValue_red,maxValue_green,maxValue_blue;
minMaxLoc(redHist, 0, &maxValue_red, 0, 0);
minMaxLoc(grayHist, 0, &maxValue_green, 0, 0);
minMaxLoc(blueHist, 0, &maxValue_blue, 0, 0);
int scale = 1;
int histHeight=256;
Mat histImage = Mat::zeros(histHeight,bins*3, CV_8UC3);

//正式开始绘制
for(int i=0;i<bins;i++)
{
    //参数准备
    float binValue_red = redHist.at<float>(i);
    float binValue_green = grayHist.at<float>(i);
    float binValue_blue = blueHist.at<float>(i);
    int intensity_red =
        cvRound(binValue_red*histHeight/maxValue_red); //要绘制的高度
    int intensity_green =
        cvRound(binValue_green*histHeight/maxValue_green); //要绘制的高度
    int intensity_blue =
        cvRound(binValue_blue*histHeight/maxValue_blue); //要绘制的高度
```

```
//绘制红色分量的直方图
rectangle(histImage, Point(i*scale,histHeight-1),
Point((i+1)*scale - 1, histHeight - intensity_red),
Scalar (255,0,0));

//绘制绿色分量的直方图
rectangle(histImage, Point((i+bins)*scale,histHeight-1),
Point((i+bins+1)*scale - 1, histHeight - intensity_green),
Scalar (0,255,0));

//绘制蓝色分量的直方图
rectangle(histImage, Point((i+bins*2)*scale,histHeight-1),
Point((i+bins*2+1)*scale - 1, histHeight - intensity_blue),
Scalar (0,0,255));

}

//在窗口中显示出绘制好的直方图
imshow( "图像的 RGB 直方图", histImage );
waitKey(0);
return 0;
}
```

运行此程序，可以得到如图 9.7、9.8 所示的结果。

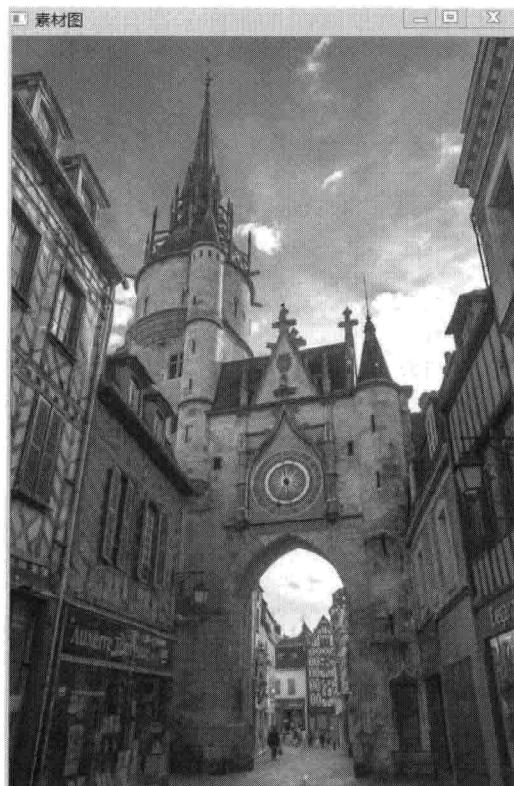


图 9.7 素材图

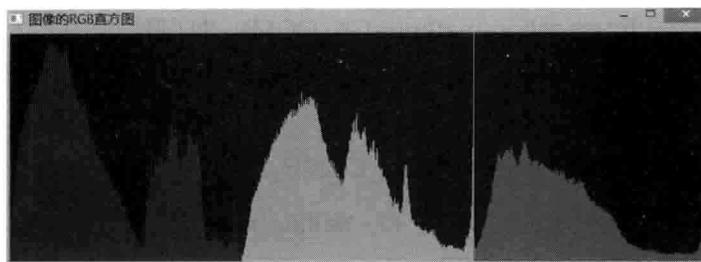


图 9.8 素材图的 RGB 直方图

至此，图像的一维和二维直方图，以及图像的三色直方图我们都通过实例进行了演示和讲解。

## 9.3 直方图对比

对于直方图来说，一个不可或缺的工具便是用某些具体的标准来比较两个直方图的相似度。要对两个直方图（比如说  $H_1$  和  $H_2$ ）进行比较，首先必须选择一个衡量直方图相似度的对比标准 ( $d(H_1, H_2)$ )。在 OpenCV 2.X 中，我们用 `compareHist()` 函数来对比两个直方图的相似度，而此函数的返回值就是  $d(H_1, H_2)$ 。

### 9.3.1 对比直方图：`compareHist()` 函数

`compareHist()` 函数用于对两幅直方图进行比较。有两个版本的 C++ 原型，如下。

```
C++: double compareHist(InputArray H1, InputArray H2, int method)
C++: double compareHist(const SparseMat& H1, const SparseMat& H2,
int method)
```

它们的前两个参数是要比较的大小相同的直方图，第三个变量是所选择的距离标准。可采用如下 4 种方法，比较两个直方图（ $H_1$  表示第一个， $H_2$  表示第二个）：

#### 1. 相关，Correlation (method=CV\_COMP\_CORREL)

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

其中：

$$\bar{H}_k = \frac{1}{N} \sum_J H_k(J)$$

且  $N$  等于直方图中 bin（译为“直条”或“组距”）的个数。

#### 2. 卡方，Chi-Square (method=CV\_COMP\_CHISQR)

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

### 3. 直方图相交, Intersection(method=CV\_COMP\_INTERSECT)

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

### 4. Bhattacharyya 距离 (method=CV\_COMP\_BHATTACHARYYA)

这里的 Bhattacharyya 距离和 Hellinger 距离相关，也可以写作 method=CV\_COMP\_HELLINGER

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{H_1 H_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

 此处的宏定义在当前版本的 OpenCV3 中依然沿用 “CV\_” 前缀，在未来版本中应该会有更改。如需使用，可以分别用 int 类型的 1、2、3、4 替代 CV\_COMP\_CORREL、CV\_COMP\_CHISQR、CV\_COMP\_INTERSECT、CV\_COMP\_BHATTACHARYYA 这四个宏。或者 “#include<cv.h>” 加入 cv.h 头文件。

## 9.3.2 示例程序：直方图对比

此次的示例程序为大家演示了如何用 compareHist() 函数进行直方图对比。代码中的 MatND 类是用于存储直方图的一种数据结构，用法简单，在这里就不多做讲解，大家看到详细注释的示例程序就会明白。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
//-----  

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;  

  
-----【main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----  

int main()
{
    //【1】声明储存基准图像和另外两张对比图像的矩阵( RGB 和 HSV )
    Mat srcImage_base, hsvImage_base;
    Mat srcImage_test1, hsvImage_test1;
    Mat srcImage_test2, hsvImage_test2;
    Mat hsvImage_halfDown;  

  
    //【2】载入基准图像(srcImage_base) 和两张测试图像srcImage_test1、
    srcImage_test2，并显示
    srcImage_base = imread("1.jpg", 1);
    srcImage_test1 = imread("2.jpg", 1);
    srcImage_test2 = imread("3.jpg", 1);
    //显示载入的3张图像
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
imshow("基准图像",srcImage_base);
imshow("测试图像 1",srcImage_test1);
imshow("测试图像 2",srcImage_test2);

// 【3】将图像由BGR色彩空间转换到 HSV 色彩空间
cvtColor( srcImage_base, hsvImage_base, COLOR_BGR2HSV );
cvtColor( srcImage_test1, hsvImage_test1, COLOR_BGR2HSV );
cvtColor( srcImage_test2, hsvImage_test2, COLOR_BGR2HSV );

// 【4】创建包含基准图像下半部的半身图像(HSV格式)
hsvImage_halfDown = hsvImage_base( Range( hsvImage_base.rows/2,
hsvImage_base.rows - 1 ), Range( 0, hsvImage_base.cols - 1 ) );

// 【5】初始化计算直方图需要的实参
// 对hue通道使用30个bin,对saturation通道使用32个bin
int h_bins = 50; int s_bins = 60;
int histSize[] = { h_bins, s_bins };
// hue的取值范围从0到256, saturation取值范围从0到180
float h_ranges[] = { 0, 256 };
float s_ranges[] = { 0, 180 };
const float* ranges[] = { h_ranges, s_ranges };
// 使用第0和第1通道
int channels[] = { 0, 1 };

// 【6】创建储存直方图的 MatND 类的实例
MatND baseHist;
MatND halfDownHist;
MatND testHist1;
MatND testHist2;

// 【7】计算基准图像,两张测试图像,半身基准图像的HSV直方图
calcHist( &hsvImage_base, 1, channels, Mat(), baseHist, 2, histSize,
ranges, true, false );
normalize( baseHist, baseHist, 0, 1, NORM_MINMAX, -1, Mat() );

calcHist( &hsvImage_halfDown, 1, channels, Mat(), halfDownHist, 2,
histSize, ranges, true, false );
normalize( halfDownHist, halfDownHist, 0, 1, NORM_MINMAX, -1,
Mat() );

calcHist( &hsvImage_test1, 1, channels, Mat(), testHist1, 2,
histSize, ranges, true, false );
normalize( testHist1, testHist1, 0, 1, NORM_MINMAX, -1, Mat() );

calcHist( &hsvImage_test2, 1, channels, Mat(), testHist2, 2,
histSize, ranges, true, false );
normalize( testHist2, testHist2, 0, 1, NORM_MINMAX, -1, Mat() );

// 【8】按顺序使用4种对比标准将基准图像的直方图与其余各直方图进行对比
for( int i = 0; i < 4; i++ )
{
```

```
//进行图像直方图的对比
int compare_method = i;
double base_base = compareHist( baseHist, baseHist,
compare_method );
double base_half = compareHist( baseHist, halfDownHist,
compare_method );
double base_test1 = compareHist( baseHist, testHist1,
compare_method );
double base_test2 = compareHist( baseHist, testHist2,
compare_method );

//输出结果
printf( " 方法 [%d] 的匹配结果如下: \n\n 【基准图 - 基准图】: %f, 【基
准图 - 半身图】: %f, 【基准图 - 测试图 1】: %f, 【基准图 - 测试图 2】: %f
\n-----\n",
i, base_base, base_half , base_test1, base_test2 );
}

printf( "检测结束。" );
waitKey(0);
return 0;
}
```

学习完详细注释的代码，让我们一起看看程序的运行截图。首先是三张素材图，如图 9.9~9.11 所示。



图 9.9 基准图像

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

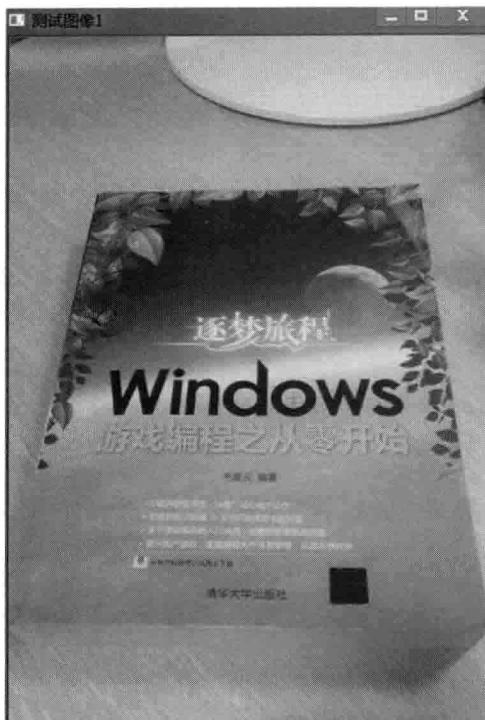


图 9.10 测试图像 (1)

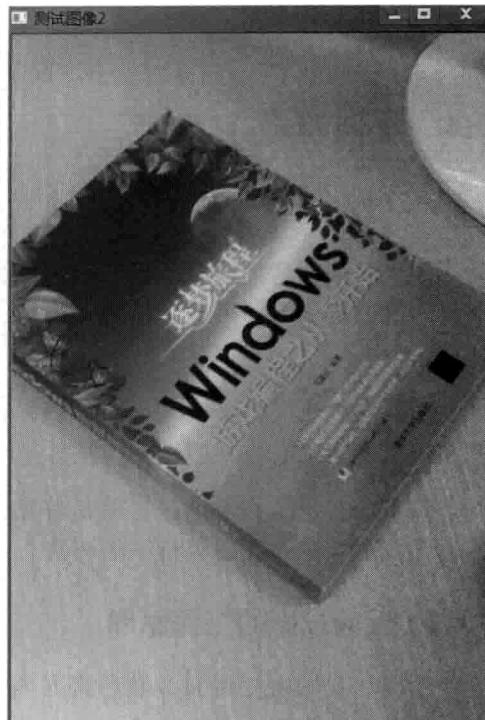


图 9.11 测试图像 (2)

需要注意的是，在上述代码中还会将基准图像与它自身及其半身图像进行对比。而我们知道，当将基准图像直方图及其自身进行对比时，会产生完美的匹配；当与来源于同一样的背景环境的半身图对比时，应该会有比较高的相似度；当与来自不同亮度光照条件的其余两张测试图像对比时，匹配度应该不是很好。输出的匹配结果如图 9.12 所示。

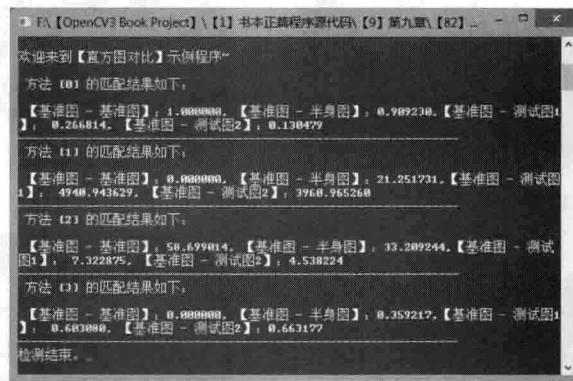


图 9.12 直方图对比结果截图

其中的方法 0 至 3，分别表示之前讲过的 Correlation、Chi-square、Intersection、Bhattacharyya 对比标准。其中，对于 Correlation（方法 0）和 Intersection（方法 2）标准，值越大表示相似度越高。可以发现，【基准图—基准图】的匹配数值结果相对于其他几种匹配方式是最大的，符合实际情况。【基准图—半身图】的匹配结果次大，正如我们预料。而【基准—测试图 1】和【基准图—测试图 2】的匹配结果

却不尽人意，同样和之前的预料吻合。

## 9.4 反向投影

### 9.4.1 引言

如果一幅图像的区域中显示的是一种结构纹理或者一个独特的物体，那么这个区域的直方图可以看作一个概率函数，其表现形式是某个像素属于该纹理或物体的概率。

而反向投影（back projection）就是一种记录给定图像中的像素点如何适应直方图模型像素分布方式的一种方法。

简单的讲，所谓反向投影就是首先计算某一特征的直方图模型，然后使用模型去寻找图像中存在的该特征的方法。

### 9.4.2 反向投影的工作原理

下面，我们将使用 H-S 肤色直方图为例来解释反向投影的工作原理。

首先通过之前讲过的求 H-S 直方图的示例程序，得到如图 9.13~9.16 所示的 H-S 肤色直方图。

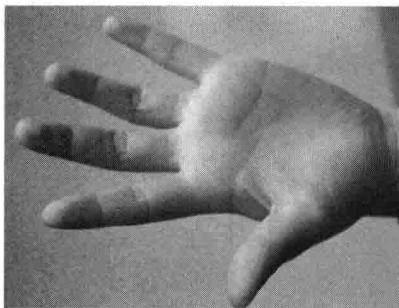


图 9.13 第一张素材图

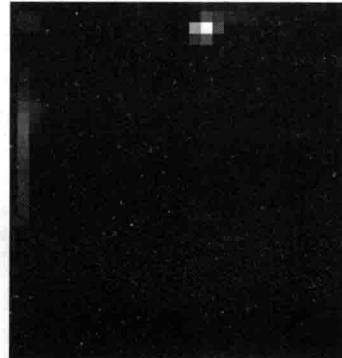


图 9.14 第一张素材图的 H-S 直方图

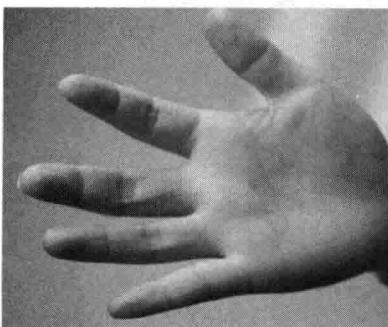


图 9.15 第二张素材图



图 9.16 第二张素材图的 H-S 直方图

而我们要做的，就是使用模型直方图（代表手掌的皮肤色调）来检测测试图像中的皮肤区域。以下是检测步骤。

(1) 对测试图像中的每个像素 ( $p(i,j)$ )，获取色调数据并找到该色调 ( $h_{i,j}, s_{i,j}$ ) 在直方图中的 bin 的位置。

(2) 查询模型直方图中对应 bin 的数值。

(3) 将此数值储存在新的反射投影图像中。也可以先归一化直方图数值到 0-255 范围，这样可以直接显示反射投影图像（单通道图像）。

(4) 通过对测试图像中的每个像素采用以上步骤，可以得到最终的反射投影图像。如图 9.17 所示。

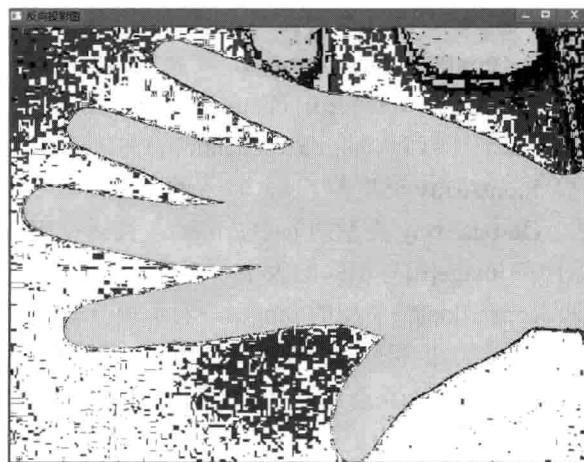


图 9.17 最终的反射投影图像

(5) 使用统计学的语言进行分析。反向投影中储存的数值代表了测试图像中该像素属于皮肤区域的概率。比如以图 9.17 为例，亮起的区域是皮肤区域的概率更大，而更暗的区域则表示是皮肤的概率更低。另外，可以注意到，手掌内部和边缘的阴影影响了检测的精度。

### 9.4.3 反向投影的作用

反向投影用于在输入图像（通常较大）中查找与特定图像（通常较小或者仅 1 个像素，以下将其称为模板图像）最匹配的点或者区域，也就是定位模板图像出现在输入图像的位置。

### 9.4.4 反向投影的结果

反向投影的结果包含了以每个输入图像像素点为起点的直方图对比结果。可以把它看成是一个二维的浮点型数组、二维矩阵，或者单通道的浮点型图像。

### 9.4.5 计算反向投影：calcBackProject()函数

calcBackProject()函数用于计算直方图的反向投影。

```
C++: void calcBackProject(
    const Mat* images,
    int nimages,
    const int* channels,
    InputArray hist,
    OutputArray backProject,
    const float** ranges,
    double scale=1,
    bool uniform=true )
```

- 第一个参数，`const Mat*`类型的`images`，输入的数组(或数组集)，它们须为相同的深度(CV\_8U 或 CV\_32F) 和相同的尺寸，而通道数则可以任意。
- 第二个参数，`int`类型的`nimages`，输入数组的个数，也就是第一个参数中存放了多少张“图像”，有几个原数组。
- 第三个参数，`const int*`类型的`channels`，需要统计的通道(dim)索引。第一个数组通道从0到`images[0].channels()-1`，而第二个数组通道从`images[0].channels()`计算到`images[0].channels() + images[1].channels()-1`。
- 第四个参数，`InputArray`类型的`hist`，输入的直方图。
- 第五个参数，`OutputArray`类型的`backProject`，目标反向投影阵列，其须为单通道，并且和`image[0]`有相同的大小和深度。
- 第六个参数，`const float**`类型的`ranges`，表示每一个维度数组(第六个参数`dims`)的每一维的边界阵列，可以理解为每一维数值的取值范围。
- 第七个参数，`double scale`，有默认值1，输出的方向投影可选的缩放因子，默认值为1。
- 第八个参数，`bool`类型的`uniform`，指示直方图是否均匀的标识符，有默认值`true`。

#### 9.4.6 通道复制：mixChannels()函数

此函数由输入参数复制某通道到输出参数特定的通道中。有两个版本的C++原型，采用函数注释方式分别介绍如下。

```
C++: void mixChannels(
    const Mat* src, //输入的数组，所有的矩阵必须有相同的尺寸和深度
    size_t nsrccs, //第一个参数 src 输入的矩阵数
    Mat* dst, //输出的数组，所有矩阵必须被初始化，且大小和深度必须与 src[0] 相同
    size_t ndsts, //第三个参数 dst 输入的矩阵数
    const int* fromTo, //对指定的通道进行复制的数组索引
    size_t npairs) //第五个参数 fromTo 的索引数
```

```
C++: void mixChannels(
    const vector<Mat>& src, //输入的矩阵向量，所有的矩阵必须有相同的尺寸和深度
    vector<Mat>& dst, //输出的矩阵向量，所有矩阵须被初始化，且大小和深度须与 src[0] 相同
    const int* fromTo, //对指定的通道进行复制的数组索引
    size_t npairs) //第三个参数 fromTo 的索引数
```

此函数为重排图像通道提供了比较先进的机制。其实，之前我们接触到的 split() 和 merge()，以及 cvtColor() 的某些形式，都只是 mixChannels() 的一部分。

下面给出一个示例，将一个 4 通道的 RGBA 图像转化为 3 通道 BGR (R 通道和 B 通道交换) 和一个单独的 Alpha 通道的图像。

```
Mat rgba( 100, 100, CV_8UC4, Scalar(1,2,3,4) );
Mat bgr( rgba.rows, rgba.cols, CV_8UC3 );
Mat alpha( rgba.rows, rgba.cols, CV_8UC1 );

//组成矩阵数组来进行操作
Mat out[] = { bgr, alpha };
// 说明: 将 rgba[0] -> bgr[2], rgba[1] -> bgr[1],
// 说明: 将 rgba[2] -> bgr[0], rgba[3] -> alpha[0]
int from_to[] = { 0,2, 1,1, 2,0, 3,3 };
mixChannels( &rgba, 1, out, 2, from_to, 4 );
```

#### 9.4.7 综合程序：反向投影

下面将给大家展示一个浓缩了本节内容的讲解内容经过详细注释的反向投影示例程序。

```
-----【头文件、命名空间包含部分】-----
//      描述: 包含程序所使用的头文件和命名空间
-----

#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
using namespace cv;

-----【宏定义部分】-----
//  描述: 定义一些辅助宏
-----

#define WINDOW_NAME1 "【原始图】"          //为窗口标题定义的宏


-----【全局变量声明部分】-----
//      描述: 全局变量声明
-----

Mat g_srcImage; Mat g_hsvImage; Mat g_hueImage;
int g_bins = 30;//直方图组距


-----【全局函数声明部分】-----
//      描述: 全局函数声明
-----

void on_BinChange(int, void* );


-----【main() 函数】-----
//      描述: 控制台应用程序的入口函数, 我们的程序从这里开始执行
-----

int main()
{
```

## Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```

//【1】读取源图像，并转换到 HSV 空间
g_srcImage = imread( "1.jpg", 1 );
if(!g_srcImage.data ) { printf("读取图片错误，请确定目录下是否有
imread函数指定图片存在~! \n"); return false; }
cvtColor( g_srcImage, g_hsvImage, COLOR_BGR2HSV );

//【2】分离 Hue 色调通道
g_hueImage.create( g_hsvImage.size(), g_hsvImage.depth() );
int ch[ ] = { 0, 0 };
mixChannels( &g_hsvImage, 1, &g_hueImage, 1, ch, 1 );

//【3】创建 Trackbar 来输入 bin 的数目
namedWindow( WINDOW_NAME1 , WINDOW_AUTOSIZE );
createTrackbar("色调组距 ", WINDOW_NAME1 , &g_bins, 180,
on_BinChange );
on_BinChange(0, 0); //进行一次初始化

//【4】显示效果图
imshow( WINDOW_NAME1 , g_srcImage );

//等待用户按键
waitKey(0);
return 0;
}

//-----【on_HoughLines() 函数】-----
//      描述：响应滑动条移动消息的回调函数
//-----
void on_BinChange(int, void* )
{
    //【1】参数准备
    MatND hist;
    int histSize = MAX( g_bins, 2 );
    float hue_range[] = { 0, 180 };
    const float* ranges = { hue_range };

    //【2】计算直方图并归一化
    calcHist( &g_hueImage, 1, 0, Mat(), hist, 1, &histSize, &ranges, true,
false );
    normalize( hist, hist, 0, 255, NORM_MINMAX, -1, Mat() );

    //【3】计算反向投影
    MatND backproj;
    calcBackProject( &g_hueImage, 1, 0, hist, backproj, &ranges, 1,
true );

    //【4】显示反向投影
    imshow( "反向投影图", backproj );

    //【5】绘制直方图的参数准备
    int w = 400; int h = 400;
}

```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
int bin_w = cvRound( (double) w / histSize );
Mat histImg = Mat::zeros( w, h, CV_8UC3 );

//【6】绘制直方图
for( int i = 0; i < g_bins; i ++ )
{ rectangle( histImg, Point( i*bin_w, h ), Point( (i+1)*bin_w,
h - cvRound( hist.at<float>(i)*h/255.0 ) ), Scalar(100, 123, 255), -1 ); }

//【7】显示直方图窗口
imshow( "直方图", histImg );
}
```

编译并运行此程序，可以通过滑动条的调节改变直方组图距（bin）的值，得到不同的反向投影效果图。如图 9.18~9.22 所示。

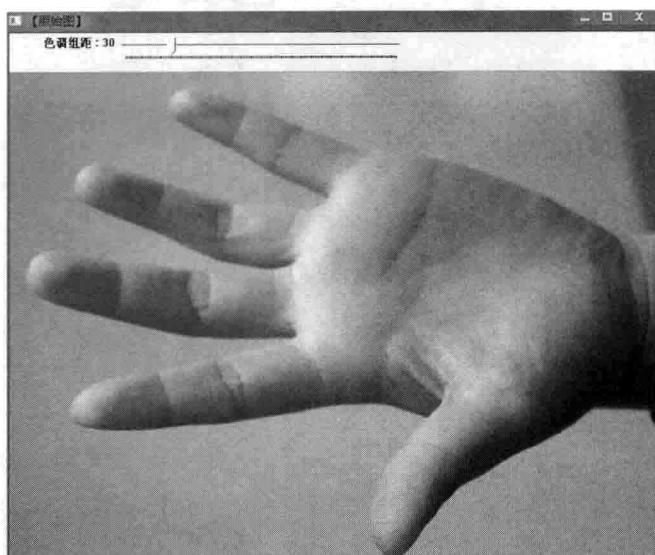


图 9.18 原始图窗口

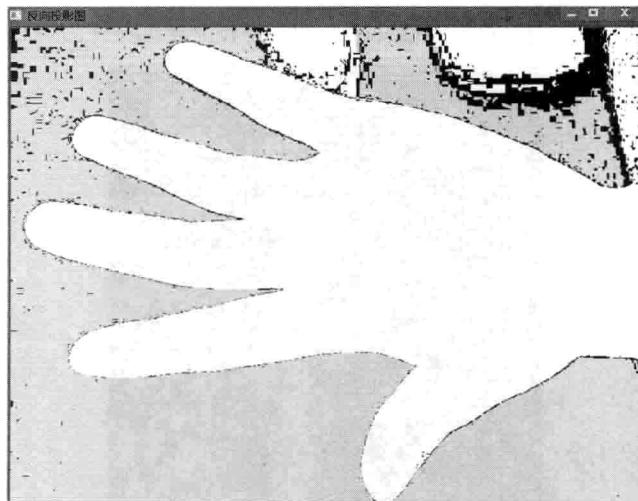


图 9.19 组距 bin 为 8 时的反向投影图

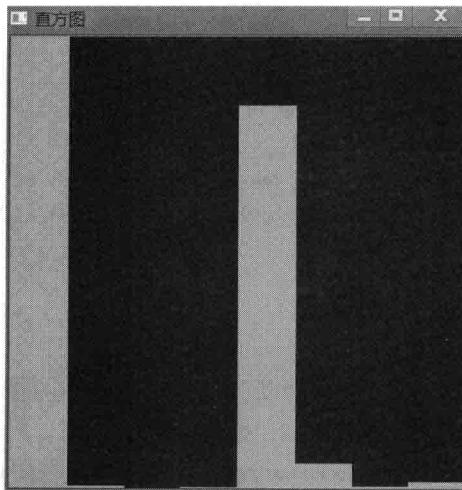


图 9.20 组距 bin 为 8 时的一维 hue 直方图



图 9.21 组距 bin 为 30 时的反向投影图

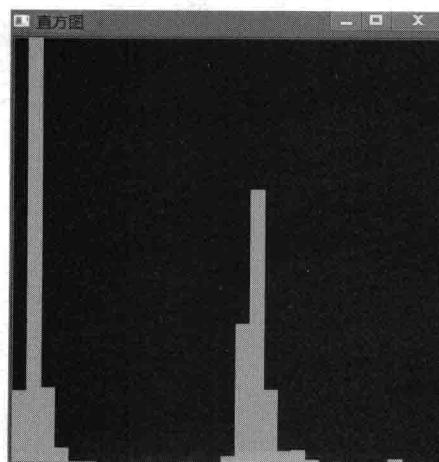


图 9.22 组距 bin 为 30 时的一维直方图

## 9.5 模板匹配

### 9.5.1 模板匹配的概念与原理

模板匹配是一项在一幅图像中寻找与另一幅模板图像最匹配（相似）部分的技术。在 OpenCV2 和 OpenCV3 中，模板匹配由 MatchTemplate() 函数完成。需要注意，模板匹配不是基于直方图的，而是通过在输入图像上滑动图像块，对实际的图像块和输入图像进行匹配的一种匹配方法。

如图 9.23 所示，通过一个人脸“图像模板”，在整幅输入图像上移动这张“脸”，来寻找和这张脸相似的最优匹配。

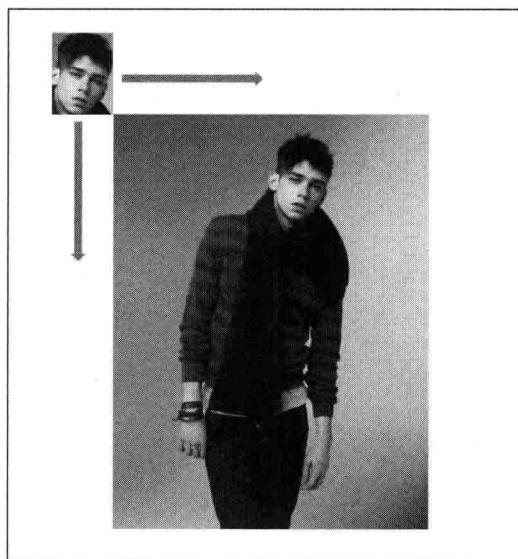


图 9.23 模板匹配图示

### 9.5.2 实现模板匹配：matchTemplate() 函数

matchTemplate() 用于匹配出和模板重叠的图像区域。

```
C++: void matchTemplate(InputArray image, InputArray templ, OutputArray result, int method)
```

- 第一个参数，InputArray 类型的 image，待搜索的图像，且需为 8 位或 32 位浮点型图像。
- 第二个参数，InputArray 类型的 templ，搜索模板，需和源图片有一样的数据类型，且尺寸不能大于源图像。
- 第三个参数，OutputArray 类型的 result，比较结果的映射图像。其必须为单通道、32 位浮点型图像。如果图像尺寸是  $W \times H$  而 templ 尺寸是  $w \times h$ ，则此参数 result 一定是  $(W-w+1) \times (H-h+1)$ 。
- 第四个参数，int 类型的 method，指定的匹配方法，OpenCV 为我们提供了如下 6 种图像匹配方法可供使用。

### 1. 平方差匹配法 method=TM\_SQDIFF

这类方法利用平方差来进行匹配，最好匹配为 0。而若匹配越差，匹配值则越大。

$$R(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

### 2. 归一化平方差匹配法 method=TM\_SQDIFF\_NORMED

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

### 3. 相关匹配法 method=TM\_CCORR

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

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

### 4. 归一化相关匹配法 method=TM\_CCORR\_NORMED

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

### 5. 系数匹配法 method=TM\_CCOEFF

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

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

其中：

$$\begin{aligned} T'(x', y') &= T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'') \\ I'(x + x', y + y') &= I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'') \end{aligned}$$

### 6. 化相关系数匹配法 method=TM\_CCOEFF\_NORMED

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



上述的 6 个宏，在 OpenCV2 依然可以加上 “CV\_” 前缀，分别写作：

CV\_TM\_SQDIFF、CV\_TM\_SQDIFF\_NORMED、CV\_TM\_CCORR、CV\_TM\_CCORR\_NORMED、CV\_TM\_CCOEFF、CV\_TM\_CCOEFF\_NORMED。

通常，随着从简单的测量（平方差）到更复杂的测量（相关系数），我们可获

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

得越来越准确的匹配。然而，这同时也会以越来越大的计算量为代价。比较科学的办法是对所有这些方法多次测试实验，以便为自己的应用选择同时兼顾速度和精度的最佳方案。

### 9.5.3 综合示例：模板匹配

讲解完基本概念和函数用法，下面依然是放出一个经过详细注释的示例程序源代码，演示了如何种不同的模板匹配方法对人脸进行检测。

```
-----【头文件与命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----  

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;  

-----【宏定义部分】-----
//  描述：定义一些辅助宏
//-----  

#define WINDOW_NAME1 "【原始图片】"          //为窗口标题定义的宏
#define WINDOW_NAME2 "【效果窗口】"          //为窗口标题定义的宏  

-----【全局变量声明部分】-----
//      描述：全局变量的声明
//-----  

Mat g_srcImage; Mat g_templateImage; Mat g_resultImage;
int g_nMatchMethod;
int g_nMaxTrackbarNum = 5;  

-----【全局函数声明部分】-----
//      描述：全局函数的声明
//-----  

void on_Matching( int, void* );  

-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----  

int main()
{
    //【1】载入原图像和模板块
    g_srcImage = imread( "1.jpg", 1 );
    g_templateImage = imread( "2.jpg", 1 );  

    //【2】创建窗口
    namedWindow( WINDOW_NAME1, CV_WINDOW_AUTOSIZE );
    namedWindow( WINDOW_NAME2, CV_WINDOW_AUTOSIZE );  

    //【3】创建滑动条并进行一次初始化
    createTrackbar( "方法", WINDOW_NAME1, &g_nMatchMethod,
    g_nMaxTrackbarNum, on_Matching );
```

```
on_Matching( 0, 0 );

waitKey(0);
return 0;

}

-----【 on_Matching() 函数 】-----
//      描述：回调函数
-----
void on_Matching( int, void* )
{
    //【1】给局部变量初始化
    Mat srcImage;
    g_srcImage.copyTo( srcImage );

    //【2】初始化用于结果输出的矩阵
    int resultImage_cols = g_srcImage.cols - g_templateImage.cols + 1;
    int resultImage_rows = g_srcImage.rows - g_templateImage.rows + 1;
    g_resultImage.create( resultImage_cols, resultImage_rows,
    CV_32FC1 );

    //【3】进行匹配和标准化
    matchTemplate( g_srcImage, g_templateImage, g_resultImage,
    g_nMatchMethod );
    normalize( g_resultImage, g_resultImage, 0, 1, NORM_MINMAX, -1,
    Mat() );

    //【4】通过函数 minMaxLoc 定位最匹配的位置
    double minValue; double maxValue; Point minLocation; Point
maxLocation;
    Point matchLocation;
    minMaxLoc( g_resultImage, &minValue, &maxValue, &minLocation,
&maxLocation, Mat() );

    //【5】对于方法 SQDIFF 和 SQDIFF_NORMED，越小的数值有着更高的匹配结果。而
其余的方法，数值越大匹配效果越好
    //此句代码的 OpenCV2 版为：
    //if( g_nMatchMethod == CV_TM_SQDIFF || g_nMatchMethod ==
CV_TM_SQDIFF_NORMED )
    //此句代码的 OpenCV3 版为：
    if( g_nMatchMethod == TM_SQDIFF || g_nMatchMethod ==
TM_SQDIFF_NORMED )
        { matchLocation = minLocation; }
    else
        { matchLocation = maxLocation; }

    //【6】绘制出矩形，并显示最终结果
    rectangle( srcImage, matchLocation, Point( matchLocation.x +
g_templateImage.cols , matchLocation.y + g_templateImage.rows ),
Scalar(0,0,255), 2, 8, 0 );
    rectangle( g_resultImage, matchLocation, Point( matchLocation.x +
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
g_templateImage.cols , matchLocation.y + g_templateImage.rows ),  
Scalar(0,0,255), 2, 8, 0 );  
  
imshow( WINDOW_NAME1, srcImage );  
imshow( WINDOW_NAME2, g_resultImage );  
  
}  
}
```

运行此程序，可以得到三个窗口。说明窗口、附有滑动条的原始图窗口以及根据滑动条数值变化的匹配效果图窗口。

首先，让我们通过说明窗口了解此程序的使用方法。如图 9.25 所示。



图 9.25 序说明窗口

接着，让我们一起看看匹配的效果图。如图 9.26~9.32 所示。

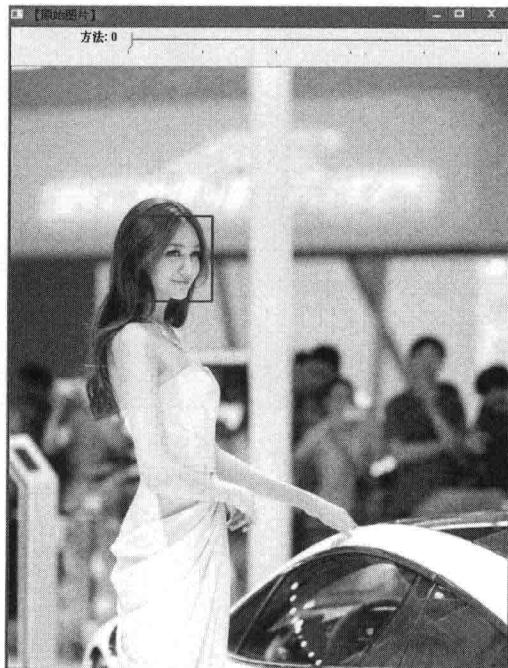


图 9.26 附有滑动条的原始图窗口

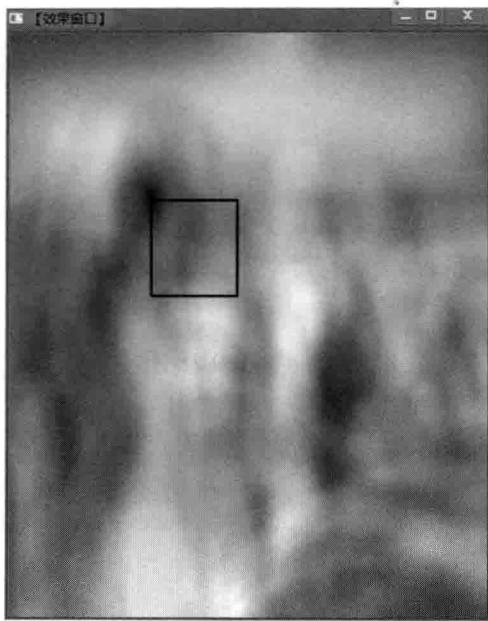


图 9.27 平方差匹配法(SQDIFF)匹配图

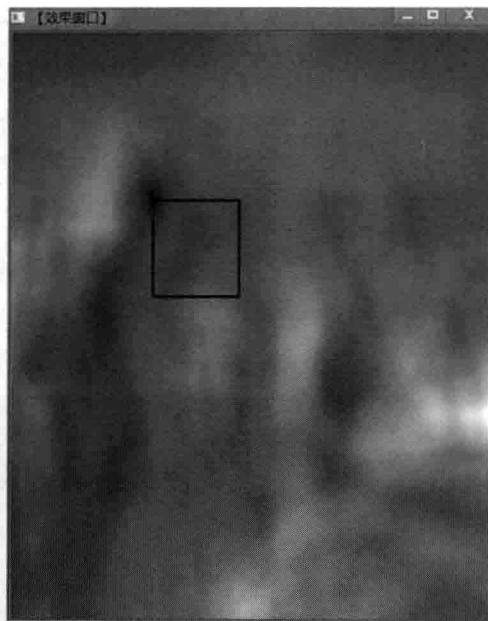


图 9.28 归一化平方差匹配法(SQDIFF NORMED)匹配图



图 9.29 相关匹配法(TM CCORR)效果图

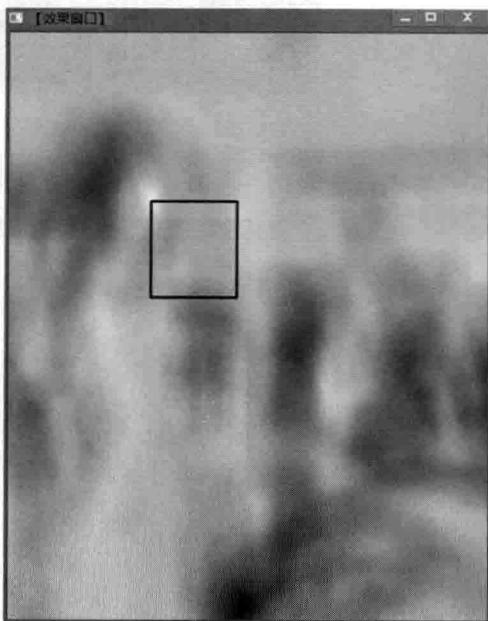


图 9.30 归一化相关匹配法(TM CCORR NORMED)

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

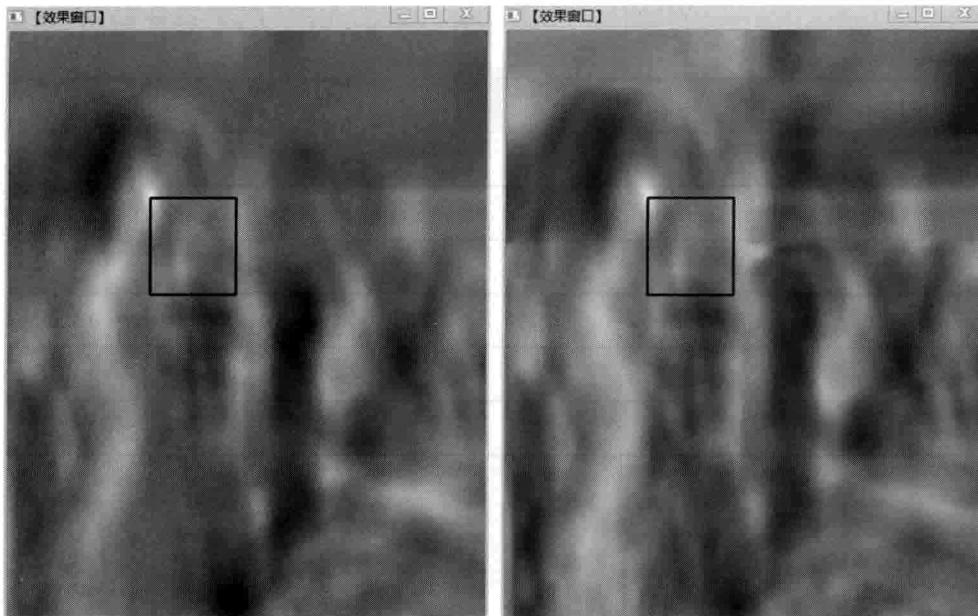


图 9.31 相关系数匹配法 (TM COEFF)

图 9.32 归一化相关系数匹配法 (TM COEFF NORMED)

可以发现，除了相关匹配法 (TM CCORR) 得到了错误的匹配结果之外，其他的 5 种匹配方式结果都匹配较为准确。

## 9.6 本章小结

本章我们学习了广泛运用于很多计算机视觉运用当中的直方图，而简单点说，直方图就是对数据进行统计的一种方法。然后还讲到了反向投影和模板匹配。所谓反向投影就是首先计算某一特征的直方图模型，最后使用模型去寻找图像中存在的该特征的方法。而模板匹配是一项在一幅图像中寻找与另一幅模板图像最匹配（相似）部分的技术。

### 本章核心函数清单

函数名称	说明	对应讲解章节
calcHist	计算一个或者多个阵列的直方图	9.2.1
minMaxLoc	在数组中找到全局最小值和最大值	9.2.2
compareHist	对两幅直方图进行比较	9.3.1
calcBackProject	计算直方图的反向投影	9.4.5
mixChannels	由输入参数复制某通道到输出参数特定的通道中	9.4.6
matchTemplate	匹配出和模板重叠的图像区域	9.5.2

**本章示例程序清单**

示例程序序号	程序说明	对应章节
79	H-S 二维直方图的绘制	9.2.3
80	一维直方图的绘制	9.2.4
81	RGB 三色直方图的绘制	9.2.5
82	直方图对比	9.3.2
83	反向投影	9.4.7
84	模板匹配	9.5.3

## 第四部分

# 深入 feature2d 组件

features2d 组件，也就是 Features 2D，是 OpenCV 的 2D 功能框架。

在 OpenCV2 中，将 features2d 组件与其余相关组件配合使用，已经达到了比较好的科研和实用效果。

自 OpenCV2 以来的众多著名的特征检测算子（如 SIFT、SURF、ORB 算子等）所依赖的稳定版的特征检测与匹配相关的核心源代码已经从官方发行的 OpenCV3 中移除，而转移到了一个名为 xfeature2d 的第三方库当中。此库还处于开发阶段，现阶段想投入使用要额外到 Github 下载，并需要进行繁琐的配置。且此库运行效果不稳定，不适合用于开发。所以，本书第 11 章的特征检测与匹配部分，将以稳定的 OpenCV2 进行讲解和代码实现。第 11 章的工程源代码仅在 OpenCV2 版的源码包中存在。第 10 章的内容依然是 OpenCV2、OpenCV3 双版本的实现和工程源码。

本部分包含如下内容：

- 特征检测和描述
- 特征检测器（Feature Detectors）通用接口
- 描述符提取器（Descriptor Extractors）通用接口
- 描述符匹配器（Descriptor Matchers）通用接口
- 通用描述符（Generic Descriptor）匹配器通用接口
- 关键点绘制函数和匹配功能绘制函数

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

# 第 10 章

## 角点检测

### 导读

---

角点检测（Corner Detection）是计算机视觉系统中用来获得图像特征的一种方法，广泛应用于运动检测、图像匹配、视频跟踪、三维建模和目标识别等领域中，也称为特征点检测。

角点通常被定义为两条边的交点，更严格地说法是，角点的局部邻域应该具有两个不同区域的不同方向的边界。而实际应用中，大多数所谓的角点检测方法检测的是拥有特定特征的图像点，而不仅仅是“角点”。这些特征点在图像中有具体的坐标，并具有某些数学特征，如局部最大或最小灰度、某些梯度特征等。

将以详细注释的代码为载体，学习如何用新版的 OpenCV 来检测角点。

### 本章你将学到：

---

- 什么是角点
- 角点检测的概念
- 用 OpenCV 实现 Harris 角点检测
- 用 OpenCV 实现 Shi-Tomasi 角点检测
- 亚像素级角点检测

## 10.1 Harris 角点检测

### 10.1.1 兴趣点与角点

在图像处理和与计算机视觉领域，兴趣点（interest points），也被称作关键点（key points）、特征点（feature points）。它被大量用于解决物体识别、图像识别、图像匹配、视觉跟踪、三维重建等一系列的问题。我们不再观察整幅图，而是选择某些特殊的点，然后对它们进行局部有的放矢地分析。如果能检测到足够多的这种点，同时它们的区分度很高，并且可以精确定位稳定的特征，那么这个方法就具有实用价值。

图像特征类型可以被分为如下三种：

- 边缘
- 角点 (感兴趣关键点)
- 斑点(Blobs)(感兴趣区域)

其中，角点是个很特殊的存在。如果某一点在任意方向的一个微小变动都会引起灰度很大的变化，那么我们就把它称之为角点。角点作为图像上的特征点，包含有重要的信息，在图像融合和目标跟踪及三维重建中有重要的应用价值。它们在图像中可以轻易地定位，同时，在人造物体场景，比如门、窗、桌等处也随处可见。因为角点位于两条边缘的交点处，代表了两个边缘变化的方向上的点，所以它们是可以精确定位的二维特征，甚至可以达到亚像素的精度。又由于其图像梯度有很高的变化，这种变化是可以用来帮助检测角点的。需要注意的是，角点与位于相同强度区域上的点不同，与物体轮廓上的点也不同，因为轮廓点难以在相同的其他物体上精确定位。

另外，关于角点的具体描述可以有如下几种：

- 一阶导数(即灰度的梯度)的局部最大所对应的像素点；
- 两条及以上边缘的交点；
- 图像中梯度值和梯度方向的变化速率都很高的点；
- 角点处的一阶导数最大，二阶导数为零，它指示了物体边缘变化不连续的方向。

### 10.1.2 角点检测

现有的角点检测算法并不是都十分的健壮。很多方法都要求有大量的训练集和冗余数据来防止或减少错误特征的出现。另外，角点检测方法的一个很重要的评价标准是其对多幅图像中相同或相似特征的检测能力，并且能够应对光照变化、图像旋转等图像变化。

在当前的图像处理领域，角点检测算法可归纳为以下三类。

- 基于灰度图像的角点检测

- 基于二值图像的角点检测
- 基于轮廓曲线的角点检测

而基于灰度图像的角点检测又可分为基于梯度、基于模板和基于模板梯度组合三类方法。其中基于模板的方法主要考虑像素领域点的灰度变化，即图像亮度的变化，将与邻点亮度对比足够大的点定义为角点。常见的基于模板的角点检测算法有 Kitchen-Rosenfeld 角点检测算法，Harris 角点检测算法、KLT 角点检测算法及 SUSAN 角点检测算法。

接下来，让我们一起来了解 harris 角点检测。

### 10.1.3 harris 角点检测

harris 角点检测是一种直接基于灰度图像的角点提取算法，稳定性高，尤其对 L 型角点检测精度高。但由于采用了高斯滤波，运算速度相对较慢，角点信息有丢失和位置偏移的现象，而且角点提取有聚簇现象。

### 10.1.4 实现 Harris 角点检测：cornerHarris()函数

cornerHarris 函数用于在 OpenCV 中运行 Harris 角点检测算子来进行角点检测。和 cornerMinEigenVal()以及 cornerEigenValsAndVecs()函数类似，cornerHarris 函数对于每一个像素  $(x,y)$  在  $\text{blockSize} \times \text{blockSize}$  邻域内，计算  $2 \times 2$  梯度的协方差矩阵  $M(x,y)$ ，接着它计算如下式子：

$$\text{dst}(x,y) = \det M^{(x,y)} - k \cdot (\text{tr } M^{(x,y)})^2$$

就可以找出输出图中的局部最大值，即找出了角点。

其函数原型和参数解析如下。

```
C++: void cornerHarris(InputArray src, OutputArray dst, int blockSize,  
int ksize, double k, int borderType=BORDER_DEFAULT )
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可，且须为单通道 8 位或者浮点型图像。
- 第二个参数，OutputArray 类型的 dst，函数调用后的运算结果存在这里，即这个参数用于存放 Harris 角点检测的输出结果，和源图片有一样的尺寸和类型。
- 第三个参数，int 类型的 blockSize，表示邻域的大小，更多详细信息在 cornerEigenValsAndVecs()中有讲到。
- 第四个参数，int 类型的 ksize，表示 Sobel() 算子的孔径大小。
- 第五个参数，double 类型的 k，Harris 参数。
- 第六个参数，int 类型的 borderType，图像像素的边界模式。注意它有默认值 BORDER\_DEFAULT。更详细的解释，参考 borderInterpolate() 函数。

讲解完这个函数，我们看一个 Harris 角点检测示例程序，其中还用到了之前讲到的 threshold 函数。

[Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com](http://www.simpopdf.com)

```
//-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所依赖的头文件和命名空间  
//-----  
#include <opencv2/opencv.hpp>  
#include <opencv2/imgproc/imgproc.hpp>  
using namespace cv;  
  
int main()  
{  
    //以灰度模式载入图像并显示  
    Mat srcImage = imread("1.jpg", 0);  
    imshow("原始图", srcImage);  
  
    //进行 Harris 角点检测找出角点  
    Mat cornerStrength;  
    cornerHarris(srcImage, cornerStrength, 2, 3, 0.01);  
  
    //对灰度图进行阈值操作，得到二值图并显示  
    Mat harrisCorner;  
    threshold(cornerStrength, harrisCorner, 0.00001, 255,  
    THRESH_BINARY);  
    imshow("角点检测后的二值效果图", harrisCorner);  
  
    waitKey(0);  
    return 0;  
}
```

运行截图如图 10.1 和图 10.2 所示。

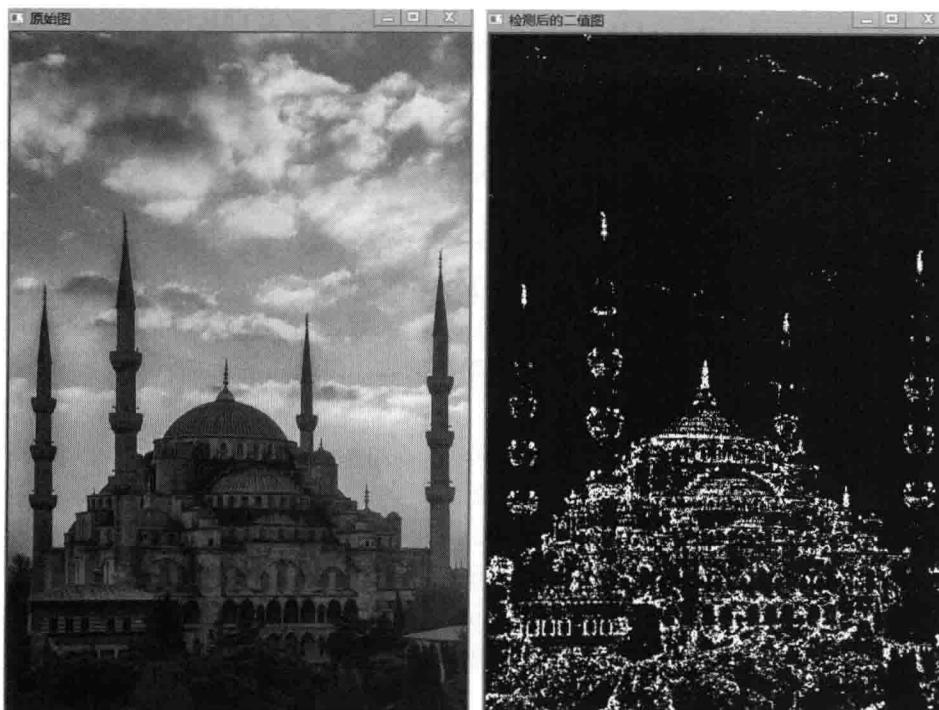


图 10.1 原始图

图 10.2 Harris 角点检测二值图

### 10.1.5 综合示例：harris 角点检测与绘制

本次综合示例为调节滚动条来控制阈值，以控制的 harris 检测角点的数量。一共有三个图片窗口，分别为显示原始图的窗口、包含滚动条的彩色效果图窗口，以及灰度图效果图窗口。

下面让我们一起来欣赏详细注释过后的完整源代码。

```
-----【头文件、命名空间包含部分】-----
//    描述：包含程序所依赖的头文件和命名空间
//-----  

#include <opencv2/opencv.hpp>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;  

-----【宏定义部分】-----
//    描述：定义一些辅助宏
//-----  

#define WINDOW_NAME1 "【程序窗口 1】"           //为窗口标题定义的宏
#define WINDOW_NAME2 "【程序窗口 2】"           //为窗口标题定义的宏  

-----【全局变量声明部分】-----
//    描述：全局变量声明
//-----  

Mat g_srcImage, g_srcImage1,g_grayImage;
int thresh = 30; //当前阈值
int max_thresh = 175; //最大阈值  

-----【全局函数声明部分】-----
//    描述：全局函数声明
//-----  

void on_CornerHarris( int, void* );//回调函数  

-----【main()函数】-----
//    描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----  

int main( int argc, char** argv )
{
    //【1】载入原始图并进行克隆保存
    g_srcImage = imread( "1.jpg", 1 );
    if(!g_srcImage.data) { printf("读取图片错误，请确定目录下是否有 imread  

函数指定的图片存在~! \n"); return false; }
    imshow("原始图",g_srcImage);
    g_srcImage1=g_srcImage.clone();  

    //【2】存留一张灰度图
    cvtColor( g_srcImage1, g_grayImage, COLOR_BGR2GRAY );
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
//【3】创建窗口和滚动条
namedWindow( WINDOW_NAME1, WINDOW_AUTOSIZE );
createTrackbar( "阈值:", WINDOW_NAME1, &thresh, max_thresh,
on_CornerHarris );

//【4】调用一次回调函数，进行初始化
on_CornerHarris( 0, 0 );

waitKey(0);
return(0);
}

-----【on_HoughLines()函数】-----
//      描述：回调函数
-----

void on_CornerHarris( int, void* )
{
    //-----【1】定义一些局部变量-----
    Mat dstImage;//目标图
    Mat normImage;//归一化后的图
    Mat scaledImage;//线性变换后的八位无符号整型的图

    //-----【2】初始化-----
    //置零当前需要显示的两幅图，即清除上一次调用此函数时他们的值
    dstImage = Mat::zeros( g_srcImage.size(), CV_32FC1 );
    g_srcImage1=g_srcImage.clone();

    //-----【3】正式检测-----
    //进行角点检测
    cornerHarris( g_grayImage, dstImage, 2, 3, 0.04, BORDER_DEFAULT );

    // 归一化与转换
    normalize( dstImage, normImage, 0, 255, NORM_MINMAX, CV_32FC1,
Mat() );
    convertScaleAbs( normImage, scaledImage );//将归一化后的图线性转换成8
位无符号整型

    //-----【4】进行绘制-----
    // 将检测到的，且符合阈值条件的角点绘制出来
    for( int j = 0; j < normImage.rows ; j++ )
    { for( int i = 0; i < normImage.cols; i++ )
    {
        if( (int) normImage.at<float>(j,i) > thresh+80 )
        {
            circle( g_srcImage1, Point( i, j ), 5, Scalar(10,10,255), 2,
8, 0 );
            circle( scaledImage, Point( i, j ), 5, Scalar(0,10,255), 2, 8,
0 );
        }
    }
}
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//-----[ 4 ]显示最终效果-----  
imshow( WINDOW_NAME1, g_srcImage1 );  
imshow( WINDOW_NAME2, scaledImage );  
  
}
```

编译并运行程序，我们可以得到如下非常漂亮的异域建筑群角点检测效果图。如图 10.3~10.7 所示。



图 10.3 原始图

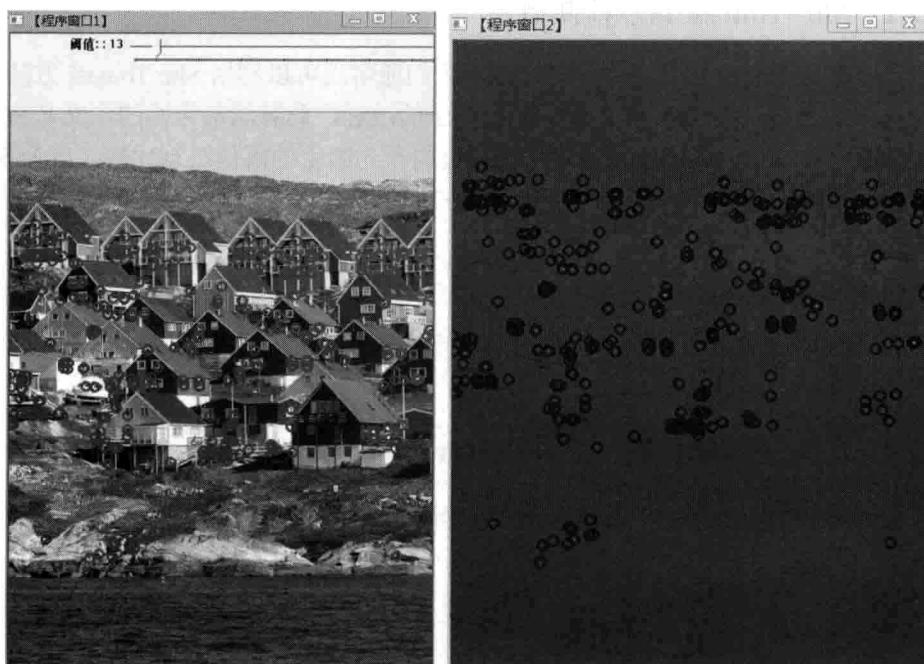


图 10.4 阈值为 13 时的检测图 (1)

图 10.5 阈值为 13 时的检测图 (2)



图 10.6 阈值为 47 时的检测图 (1)

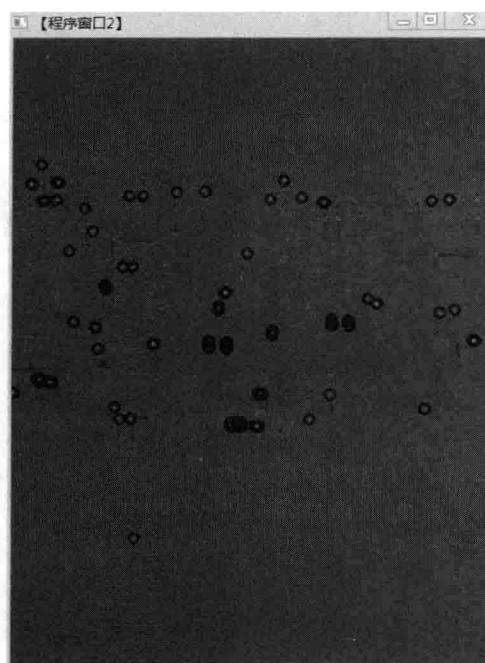


图 10.7 阈值为 47 时的检测图 (2)

## 10.2 Shi-Tomasi 角点检测

### 10.2.1 Shi-Tomasi 角点检测概述

除了利用 Harris 进行角点检测之外，我们通常还可以利用 Shi-Tomasi 方法进行角点检测。Shi-Tomasi 算法是 Harris 算法的改进，此算法最原始的定义是将矩阵  $M$  的行列式值与  $M$  的迹相减，再将差值同预先给定的阈值进行比较。后来 Shi 和 Tomasi 提出改进了方法，若两个特征值中较小的一个大于最小阈值，则会得到强角点。

由于 Shi-Tomasi 算子是 1994 年在文章《Good Features to Track》中被提出的，OpenCV 实现此算法的函数名便定义为 `goodFeaturesToTrack`。下面我们来看看此函数的用法。

### 10.2.2 确定图像强角点：`goodFeaturesToTrack()` 函数

`goodFeaturesToTrack()` 函数结合了 Shi-Tomasi 算子，用于确定图像的强角点。

```
C++: void goodFeaturesToTrack(  
    InputArray image,  
    OutputArray corners,  
    int maxCorners,
```

```
double qualityLevel,  
double minDistance,  
InputArray mask=noArray(),  
int blockSize=3,  
bool useHarrisDetector=false,  
double k=0.04 )
```

- 第一个参数，InputArray 类型的 image，输入图像，须为 8 位或浮点型 32 位单通道图像。
- 第二个参数，OutputArray 类型的 corners，检测到的角点的输出向量。
- 第三个参数，int 类型的 maxCorners，角点的最大数量。
- 第四个参数，double 类型的 qualityLevel，角点检测可接受的最小特征值。其实实际用于过滤角点的最小特征值是 qualityLevel 与图像中最大特征值的乘积。所以 qualityLevel 通常不会超过 1 (常用的值为 0.10 或者 0.01)。而检测完所有的角点后，还要进一步剔除掉一些距离较近的角点。
- 第五个参数，double 类型的 minDistance，角点之间的最小距离，此参数用于保证返回的角点之间的距离不小于 minDistance 个像素。
- 第六个参数，InputArray 类型的 mask，可选参数，表示感兴趣区域，有默认值 noArray()。若此参数非空(需为 CV\_8UC1 类型，且和第一个参数 image 有相同的尺寸)，便用于指定角点检测区域。
- 第七个参数，int 类型的 blockSize，有默认值 3，是计算导数自相关矩阵时指定的邻域范围。
- 第八个参数，bool 类型的 useHarrisDetector，默认值 false，指示是否使用 Harris 角点检测。
- 第九个参数，double 类型的 k，有默认值 0.04，为用于设置 Hessian 自相关矩阵行列式的相对权重的权重系数。

另外值得一提的是，goodFeaturesToTrack 函数可用来初始化一个基于点的对象跟踪操作。

### 10.2.3 综合示例：Shi-Tomasi 角点检测

作为一本编程入门教程，依然是发挥我们的特色——用详细注释，条理清晰的代码说话。下面就将放出一个详细注释的以 goodFeaturesToTrack 函数为核心，进行 Shi-Tomasi 角点检测的示例程序。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所使用的头文件和命名空间  
-----  
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"  
#include <iostream>  
using namespace cv;  
using namespace std;  
  
-----【宏定义部分】-----
```



Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
g_rng.uniform(0,255),  
    g_rng.uniform(0,255)), -1, 8, 0 );  
}  
  
//【6】显示(更新)窗口  
imshow( WINDOW_NAME, copy );  
}  
  
-----【main()函数】-----  
//      描述: 控制台应用程序的入口函数, 我们的程序从这里开始执行  
-----  
int main()  
{  
    //【1】载入源图像并将其转换为灰度图  
    g_srcImage = imread("1.jpg", 1 );  
    cvtColor( g_srcImage, g_grayImage, COLOR_BGR2GRAY );  
  
    //【2】创建窗口和滑动条, 并进行显示和回调函数初始化  
    namedWindow( WINDOW_NAME, WINDOW_AUTOSIZE );  
    createTrackbar( "最大角点数", WINDOW_NAME, &g_maxCornerNumber,  
    g_maxTrackbarNumber, on_GoodFeaturesToTrack );  
    imshow( WINDOW_NAME, g_srcImage );  
    on_GoodFeaturesToTrack( 0, 0 );  
  
    waitKey(0);  
    return(0);  
}
```

运行此程序，得到如图 10.8 所示的角点检测图，并且调节滑动条，可改变能检测到的最大角点数量。如图 10.9 和 10.10 所示。在程序代码中，我们设定可检测到的最大角点数量为 500。

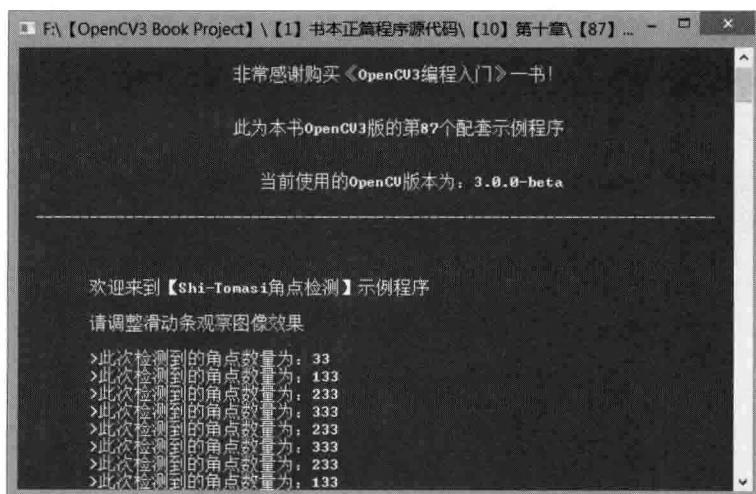


图 10.8 文本信息输出窗口



图 10.9 最大检测角点数为 33 时程序截图 图 10.10 最大检测角点数为 500 时的程序截图

## 10.3 亚像素级角点检测

### 10.3.1 背景概述

若我们进行图像处理的目的不是提取用于识别的特征点而是进行几何测量，这通常需要更高的精度，而函数 `goodFeaturesToTrack()` 只能提供简单的像素的坐标值，也就是说，有时候会需要实数坐标值而不是整数坐标值。

亚像素级角点检测的位置在摄像机标定、跟踪并重建摄像机的轨迹，或者重建被跟踪目标的三维结构时，是一个基本的测量值。

下面我们将讨论如何将所求得的角点位置精确到亚像素级精度。一个向量和与其正交的向量的点积为 0，角点则满足如图 10.11 所示情况。

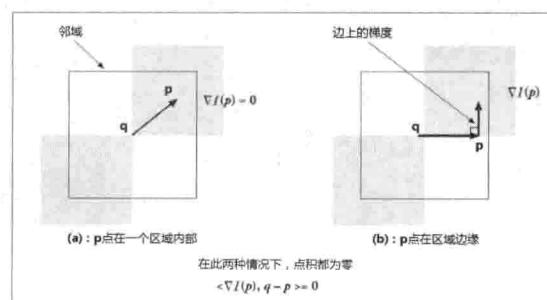


图 10.11 计算亚像素级精度的角点

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

其中，(a) 点 p 附近的图像是均匀的，其梯度为 0；(b) 边缘的梯度与沿边缘方向的 q-p 向量正交。在图中的两种情况下，p 点梯度与 q-p 向量的点积均为 0。

图 10.11 中，我们假设起始角点 q 在实际亚像素级角点的附近。检测所有的 q-p 向量。若点 p 位于一个均匀的区域，则点 p 处的梯度为 0。若 q-p 向量的方向与边缘的方向一致，则此边缘上 p 点处的梯度与 q-p 向量正交，在这两种情况下，p 点处的梯度与 q-p 向量的点积为 0。我们可以在 p 点周围找到很多组梯度以及相关的向量 q-p，令其点集为 0，然后可以通过求解方程组，方程组的解即为角点 q 的亚像素级精度的位置，也就是精确的角点位置。

OpenCV 为我们提供了 cornerSubPix() 函数，用于发现亚像素精度的角点位置。

### 10.3.2 寻找亚像素角点：cornerSubPix()函数

cornerSubPix 函数用于寻找亚像素角点位置（不是整数类型的位置，而是更精确的浮点类型位置）。

```
C++: void cornerSubPix(  
    InputArray image,  
    InputOutputArray corners,  
    Size winSize,  
    Size zeroZone,  
    TermCriteria criteria)
```

- 第一个参数，InputArray 类型的 image，输入图像，即源图像。
- 第二个参数，InputOutputArray 类型的 corners，提供输入角点的初始坐标和精确的输出坐标。
- 第三个参数，Size 类型的 winSize，搜索窗口的一半尺寸。若  $\text{winSize} = \text{Size}(5,5)$ ，那么就表示使用  $(5*2+1) \times (5*2+1) = 11 \times 11$  大小的搜索窗口。
- 第四个参数，Size 类型的 zeroZone，表示死区的一半尺寸。而死区为不对搜索区的中央位置做求和运算的区域，用来避免自相关矩阵出现的某些可能的奇异性。值为 (-1,-1) 表示没有死区。
- 第五个参数，TermCriteria 类型的 criteria，求角点的迭代过程的终止条件。即角点位置的确定，要么迭代数大于某个设定值，或者是精确度达到某个设定值。criteria 可以是最大迭代数目，或者是设定的精确度，也可以是它们的组合。

### 10.3.3 综合示例：亚像素级角点检测

这个程序在上一小节 Shi-Tomasi 角点检测示例程序的基础上，在 on\_GoodFeaturesToTrack() 回调函数中加上以下进行亚像素角点检测的代码，就成为了亚像素级角点检测的示例程序，需添加的代码如下。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

-----【 on_GoodFeaturesToTrack() 函数】-----
//      描述：响应滑动条移动消息的回调函数
//-----
void on_GoodFeaturesToTrack( int, void* )
{
    //......

    //【7】亚像素角点检测的参数设置
    Size winSize = Size( 5, 5 );
    Size zeroZone = Size( -1, -1 );
    //此句代码的OpenCV2 版为：
    //TermCriteria criteria = TermCriteria( CV_TERMCRIT_EPS +
    CV_TERMCRIT_ITER, 40, 0.001 );
    //此句代码的OpenCV3 版为：
    TermCriteria criteria = TermCriteria( TermCriteria::EPS +
    TermCriteria::MAX_ITER, 40, 0.001 );

    //【8】计算出亚像素角点位置
    cornerSubPix( g_grayImage, corners, winSize, zeroZone, criteria );

    //【9】输出角点信息
    for( int i = 0; i < corners.size(); i++ )
    {
        cout<<
        " \t>>精确角点坐["<<i<<" ] ("<<corners[i].x<<","<<corners[i].y<<" ) "
        <<endl;
    }
}

```

运行此程序，得到的窗口和之前的【Shi-Tomasi 角点检测】一致，区别在于在控制台窗口中输出了检测到的浮点型亚像素角点精确坐标值。如图 10.12~10.15 所示。

```

>>精确角点坐标[109] (182.101, 426.347, 509.625)
>>精确角点坐标[110] (273.694, 222.386)
>>精确角点坐标[111] (426.314)
>>精确角点坐标[112] (157.531)
>>精确角点坐标[113] (297.532)
>>精确角点坐标[114] (330.289, 606.427)
>>精确角点坐标[115] (422.52, 290.007)
>>精确角点坐标[116] (463.572)
>>精确角点坐标[117] (163.228, 110.751)
>>精确角点坐标[118] (249.595)
>>精确角点坐标[119] (423.883, 596.544)
>>精确角点坐标[120] (204.298, 598.317)
>>精确角点坐标[121] (212.878, 144.102)
>>精确角点坐标[122] (140.605, 604.42)
>>精确角点坐标[123] (31.2509, 530.482)
>>精确角点坐标[124] (228.481, 580.77)
>>精确角点坐标[125] (227.015, 235.797)
>>精确角点坐标[126] (391.898, 544.515)
>>精确角点坐标[127] (227.141, 627.764)
>>精确角点坐标[128] (195.978, 102.605)
>>精确角点坐标[129] (140.857, 410.413)
>>精确角点坐标[130] (190.825, 389.456)
>>精确角点坐标[131] (163.789, 468.96)

```

图 10.12 亚像素角点信息输出窗口（最大角点数为 133 时）

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

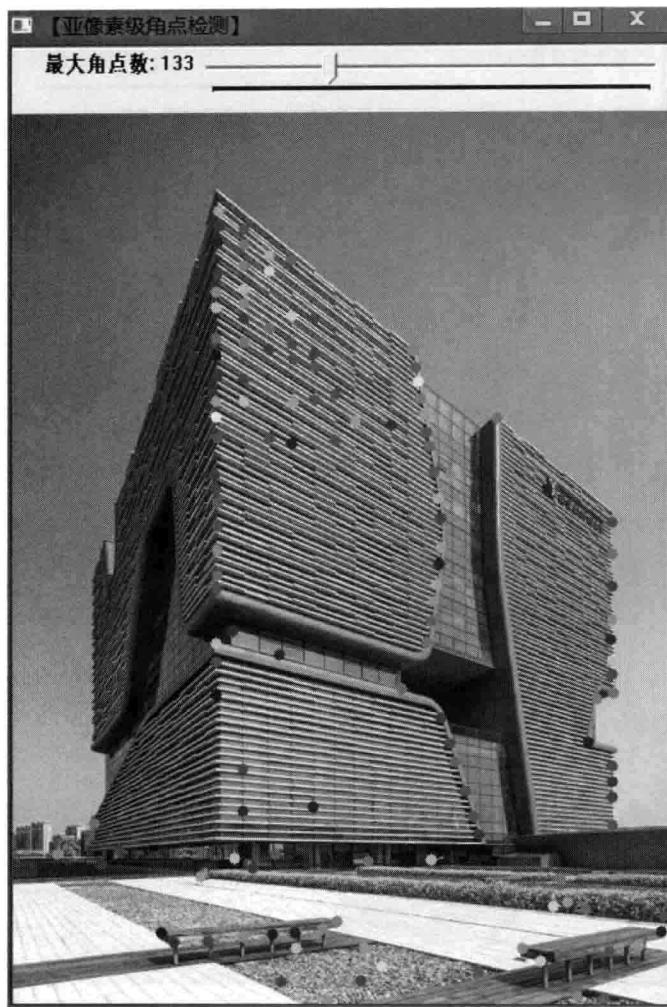


图 10.13 亚像素角点检测窗口（最大角点数为 133 时）

```
F:\Book Projects\1\亚像素级角点检测\Debug\亚像素级的角点检测.exe
>>> 精确角点坐标[277] <126, 528>
>>> 精确角点坐标[278] <307, 248, 601, 371>
>>> 精确角点坐标[279] <354, 365>
>>> 精确角点坐标[280] <159, 976, 408, 094>
>>> 精确角点坐标[281] <222, 617>
>>> 精确角点坐标[282] <321, 551>
>>> 精确角点坐标[283] <218, 035, 552, 528>
>>> 精确角点坐标[284] <241, 957, 276, 114>
>>> 精确角点坐标[285] <366, 555>
>>> 精确角点坐标[286] <364, 321>
>>> 精确角点坐标[287] <161, 658, 425, 556>
>>> 精确角点坐标[288] <58, 517>
>>> 精确角点坐标[289] <444, 571>
>>> 精确角点坐标[290] <320, 273, 603, 732>
>>> 精确角点坐标[291] <402, 281>
>>> 精确角点坐标[292] <242, 845, 495, 239>
>>> 精确角点坐标[293] <56, 4908, 350, 561>
>>> 精确角点坐标[294] <173, 64, 571, 354>
>>> 精确角点坐标[295] <227, 058, 128, 932>
>>> 精确角点坐标[296] <228, 510>
>>> 精确角点坐标[297] <232, 825, 310, 105>
>>> 精确角点坐标[298] <341, 874, 396, 034>
>>> 精确角点坐标[299] <347, 179, 556, 628>
>>> 精确角点坐标[300] <131, 231>
```

The screenshot shows a terminal window displaying a list of 301 precise corner point coordinates, each consisting of a label (e.g., '>>> 精确角点坐标[277]') followed by a coordinate pair (e.g., '<126, 528>'). The coordinates are listed vertically, representing the output of the corner detection algorithm.

图 10.14 亚像素角点信息输出窗口（最大角点数为 301 时）

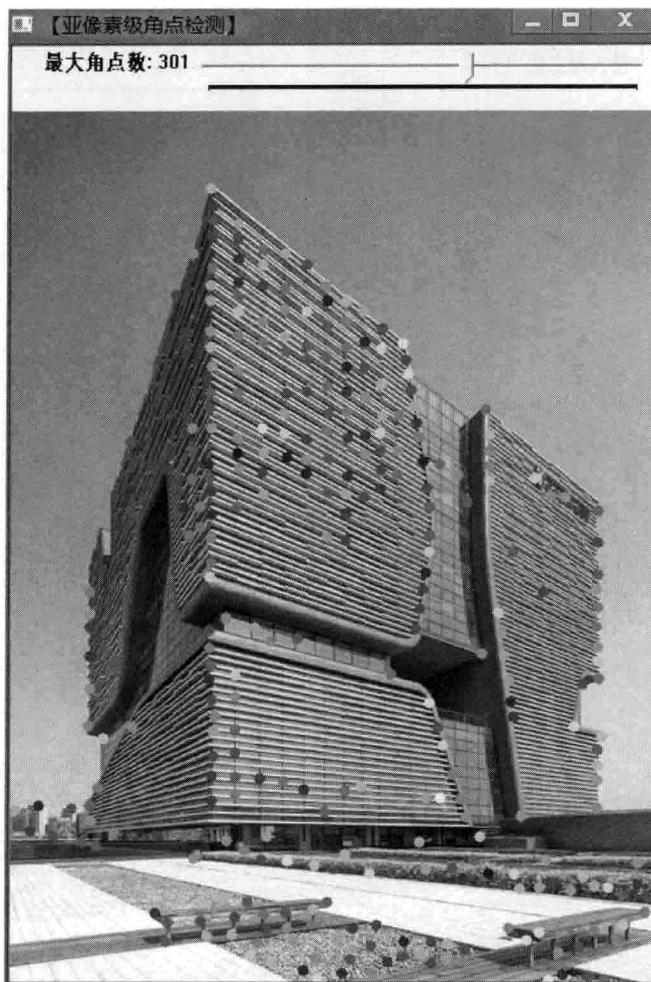


图 10.15 亚像素角点检测窗口（最大角点数为 301 时）

## 10.4 本章小结

本章我们讲解了 Harris 角点检测和 Shi-Tomasi 角点检测，以及一种亚像素角点检测方法。当然，也可以自己制作角点检测的函数，需要用到 `cornerMinEigenVal` 函数和 `minMaxLoc` 函数。最后的特征点选取，判断条件要根据自己的情况编辑。如果对特征点，角点的精度要求更高，可以用 `cornerSubPix` 函数将角点定位到子像素。

### 本章核心函数清单

函数名称	说明	对应讲解章节
<code>cornerHarris</code>	运行 Harris 角点检测算子来进行角点检测	10.1.4
<code>goodFeaturesToTrack</code>	结合 Shi-Tomasi 算子确定图像的强角点	10.2.2
<code>cornerSubPix</code>	寻找亚像素角点位置	10.3.2

**本章示例程序清单**

示例程序序号	程序说明	对应章节
85	实现 Harris 角点检测: cornerHarris()函数的使用	10.1.4
86	harris 角点检测与绘制	10.1.5
87	Shi-Tomasi 角点检测	10.2.3
88	亚像素级角点检测	10.3.3

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

# 第 11 章

## 特征检测与匹配

### 导读

---

特征点的检测和匹配是计算机视觉中非常重要的技术之一。在物体检测、视觉跟踪、三维重建等领域都有很广泛的应用。

OpenCV 为我们提供了以下 10 种特征检测方法（破折号后面为对应的类名）。

- “FAST” ——FastFeatureDetector
- “STAR” ——StarFeatureDetector
- “SIFT” ——SIFT (nonfree module)
- “SURF” ——SURF (nonfree module)
- “ORB” ——ORB
- “MSER” ——MSER
- “GFTT” ——GoodFeaturesToTrackDetector
- “HARRIS” ——GoodFeaturesToTrackDetector (配合 Harris detector 操作)
- “Dense” ——DenseFeatureDetector
- “SimpleBlob” ——SimpleBlobDetector

其中，Harris 方法在上一章中已经详细讲解过。而在本章中，我们会对最常用 SIFT、SURF 和 ORB 等算法用 OpenCV2 进行说明、讲解以及编程实现。



需要再次说明，由于目前发行的 OpenCV3 中，众多著名的特征检测算子（如 SIFT、SURF、ORB 算子等）所依赖的稳定版的源代码已经从官方发行的 OpenCV3 中移除，而转移到一个名为 xfeature2d 的第三方库当中。所以本章内容将以稳定的 OpenCV2 版本进行讲解和代码实现，本章的工程源代码也仅在 OpenCV2 版的源码包中存在。

## 11.1 SURF 特征点检测

### 11.1.1 SURF 算法概览

SURF，英文全称为 SpeededUp Robust Features，直译为“加速版的具有鲁棒性的特征”算法，由 Bay 在 2006 年首次提出。SURF 是尺度不变特征变换算法(SIFT 算法)的加速版。一般来说，标准的 SURF 算子比 SIFT 算子快好几倍，并且在多幅图片下具有更好的稳定性。SURF 最大的特征在于采用了 harr 特征以及积分图像的概念，这大大加快了程序的运行时间。SURF 可以应用于计算机视觉的物体识别以及 3D 重构中。

### 11.1.2 SURF 算法原理

#### 1. 构建 Hessian 矩阵构造高斯金字塔尺度空间

其实 surf 构造的金字塔图像与 sift 有很大不同，正是因为这些不同才加快了其检测的速度。Sift 采用的是 DOG 图像，而 surf 采用的是 Hessian 矩阵行列式近似值图像。Hessian 矩阵是 Surf 算法的核心。在数学中，海森矩阵 (Hessian matrix 或 Hessian) 是一个自变量为向量的实值函数的二阶偏导数组成的方块矩阵。为了方便运算，假设函数  $f(x, y)$ ，Hessian 矩阵  $H$  是函数、偏导数组成。首先来看看图像中某个像素点的 Hessian 矩阵，如下：

$$H(f(x, y)) = \begin{bmatrix} \frac{\partial_2 f}{\partial x^2} & \frac{\partial_2 f}{\partial x \partial y} \\ \frac{\partial_2 f}{\partial x \partial y} & \frac{\partial_2 f}{\partial y^2} \end{bmatrix}$$

即每一个像素点都可以求出一个 Hessian 矩阵。

$H$  矩阵判别式为：

$$\det(H) = \frac{\partial^2 f \partial^2 f}{\partial x^2 \partial y^2} - \left( \frac{\partial^2 f}{\partial x \partial y} \right)^2$$

判别式的值是  $H$  矩阵的特征值，可以利用判定结果的符号将所有点分类，根据判别式取值正负，来判别该点是或不是极值点。

在 SURF 算法中，用图像像素  $l(x, y)$  即为函数值  $f(x, y)$ ，选用二阶标准高斯函数作为滤波器，通过特定核间的卷积计算二阶偏导数，这样便能计算出  $H$  矩阵的三个矩阵元素  $L_{xx}$ 、 $L_{xy}$ 、 $L_{yy}$ ，从而计算出  $H$  矩阵：

$$H(x, \sigma) = \begin{bmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{bmatrix}$$

但是由于特征点需要具备尺度无关性，所以在进行 Hessian 矩阵构造前，需

要对其进行高斯滤波。

这样，经过滤波后再进行 Hessian 的计算，其公式如下：

$$L(x,t) = G(t) \cdot I(x,t)$$

$L(x,t)$ 是一幅图像在不同解析度下的表示，可以利用高斯核  $G(t)$ 与图像函数  $I(x)$  在点  $x$  的卷积来实现，其中高斯核  $G(t)$ 的计算公式：

$$G(t) = \frac{\partial^2 g(t)}{\partial x^2}$$

$g(x)$ 为高斯函数， $t$  为高斯方差。通过这种方法可以为图像中每个像素计算出其 H 行列式的决定值，并用这个值来判别特征点。为方便应用，Herbert Bay 提出用近似值先代替  $L(x,t)$ 。为平衡准确值与近似值间的误差引入权值，权值随尺度变化，则 H 矩阵判别式可表示为：

$$\det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2$$

其中 0.9 是论文作者给出的一个经验值，其实它是有一套理论计算的，具体请参考 surf 的英文论文。

由于求 Hessian 时要先高斯平滑，然后求二阶导数，这在离散的像素点是用模板卷积形成的，这 2 中操作合在一起用一个模板代替就可以了，比如说 y 方向上的模板如图 11.1 所示。

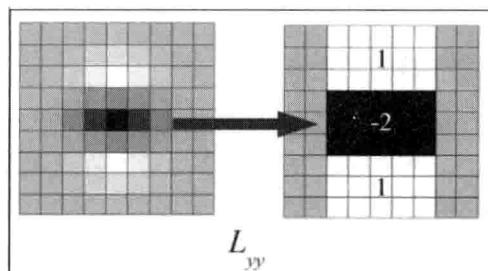


图 11.1 y 方向上的模板

图 11.1 的左边即用高斯平滑然后在 y 方向上求二阶导数的模板，为了加快运算用了近似处理，其处理结果如下图所示，这样就简化了很多。并且右图可以采用积分图来运算，从而大大加快了速度，关于积分图的介绍，可以去查阅相关的资料。

同理，x 和 y 方向的二阶混合偏导模板如图 11.2 所示。

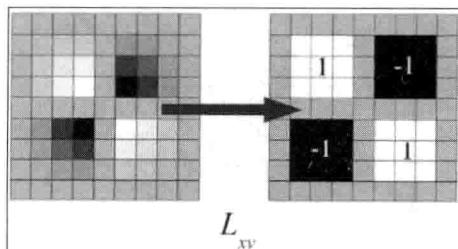


图 11.2 x 和 y 方向的二阶混合偏导模板

上面讲的这么多，只是得到了一张近似 hessian 的行列式图，这类似 sift 中的 DOG 图。但是在金字塔图像中分为很多层，每一层叫做一个 octave，每一个 octave 中又有几张尺度不同的图片。在 sift 算法中，同一个 octave 层中的图片尺寸（即大小）相同，但是尺度（即模糊程度）不同，而不同的 octave 层中的图片尺寸大小也不相同，因为它是上一层图片降采样得到的。在进行高斯模糊时，sift 的高斯模板大小是始终不变的，只是在不同的 octave 之间改变图片的大小。而在 surf 中，图片的大小是一直不变的，不同 octave 层的待检测图片是改变高斯模糊尺寸大小得到的，当然了，同一个 octave 中不同图片用到的高斯模板尺度也不同。算法允许尺度空间多层图像同时被处理，不需对图像进行二次抽样，从而提高算法性能。图 11.3 是传统方式建立的一个金字塔结构，图像的尺寸是变化的，并且运算会反复使用高斯函数对子层进行平滑处理，说明 Surf 算法使原始图像保持不变而只改变了滤波器大小。Surf 采用这种方法节省了降采样过程，其处理速度自然也就提上去了。

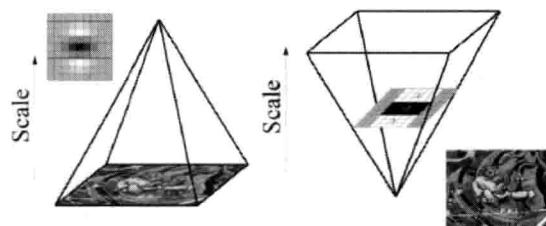


图 11.3 相关金字塔结构

## 2. 利用非极大值抑制初步确定特征点

此步骤和 sift 类似，将经过 hessian 矩阵处理过的每个像素点与其三维领域的 26 个点进行大小比较，如果它是这 26 个点中的最大值或者最小值，则保留下，当做初步的特征点。检测过程中使用与该尺度层图像解析度相对应大小的滤波器进行检测。以  $3 \times 3$  的滤波器为例，该尺度层图像中 9 个像素点之一的检测特征点与自身尺度层中其余 8 个点和在其之上及之下的两个尺度层的各 9 个点进行比较，共 26 个点，图 11.4 中标记的“x”的像素点的特征值若大于周围像素，则可确定该点为该区域的特征点。

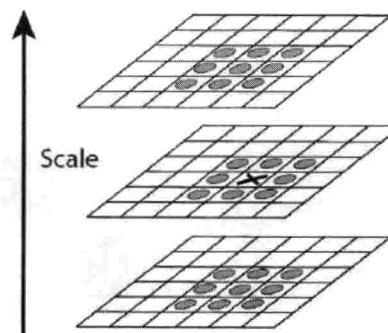


图 11.4 初步确定特征点

### 3. 精确定位极值点

这里也和 sift 算法中的类似，采用三维线性插值法得到亚像素级的特征点，同时也去掉那些值小于一定阈值的点，增加极值使检测到的特征点数量减少，最终只有几个特征最强点会被检测出来。

### 4. 选取特征点的主方向

这一步与 sift 也大有不同。Sift 选取特征点主方向是采用在特征点领域内统计其梯度直方图，取直方图 bin 值最大的以及超过最大 bin 值 80% 的那些方向做为特征点的主方向。

而在 surf 中，不统计其梯度直方图，而是统计特征点领域内的 haar 小波特征。即在特征点的领域（比如说，半径为  $6s$  的圆内， $s$  为该点所在的尺度）内，统计 60 度扇形内所有点的水平 haar 小波特征和垂直 haar 小波特征总和，haar 小波的尺寸变长为  $4s$ ，这样一个扇形得到了一个值。然后 60 度扇形以一定间隔进行旋转，最后将最大值那个扇形的方向作为该特征点的主方向。该过程的示意图如图 11.5 所示。

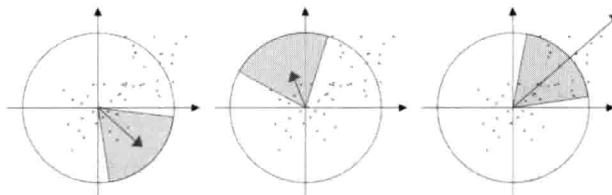


图 11.5 选取特征点的主方向

### 5. 构造 surf 特征点描述算子

在 sift 中，是在特征点周围取  $16 \times 16$  的邻域，并把该领域化为  $4 \times 4$  个的小区域，每个小区域统计 8 个方向梯度，最后得到  $4 \times 4 \times 8 = 128$  维的向量，该向量作为该点的 sift 描述子。

在 surf 中，也是在特征点周围取一个正方形框，框的边长为  $20s$  ( $s$  是所检测到该特征点所在的尺度)。该框带方向，方向当然就是第 4 步检测出来的主方向了。然后把该框分为 16 个子区域，每个子区域统计 25 个像素的水平方向和垂直方向的 haar 小波特征，这里的水平和垂直方向都是相对主方向而言的。该 haar 小波特征为水平方向值之和，水平方向绝对值之和，垂直方向之和，垂直方向绝对值之和。该过程的示意图如图 11.6 所示。

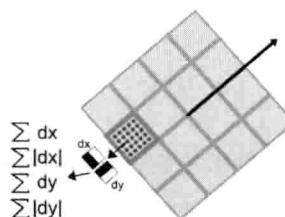


图 11.6 构造 surf 特征点描述算子

[Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com](http://www.simpopdf.com)

这样每个小区域就有 4 个值，所以每个特征点就是  $16 \times 4 = 64$  维的向量，相比 sift 而言，少了一半，这在特征匹配过程中会大大加快匹配速度。

## 6. 总结

Surf 采用 Hessian 矩阵获取图像局部最值十分稳定，但是在求主方向阶段太过于依赖局部区域像素的梯度方向，有可能使找到的主方向不准确。后面的特征向量提取以及匹配都严重依赖于主方向，即使不大偏差角度也可以造成后面特征匹配的放大误差，从而使匹配不成功。另外图像金字塔的层取得不够紧密也会使得尺度有误差，后面的特征向量提取同样依赖相应的尺度，发明者在这个问题上的折中解决方法是取适量的层然后进行插值。

### 11.1.3 SURF 类相关 OpenCV 源码剖析

OpenCV 中关于 SURF 算法的部分，常常涉及到的是 SURF、SurfFeatureDetector、SurfDescriptorExtractor 这三个类，这一小节我们就来看看它们究竟是什么来头。

在……\opencv\sources\modules\nonfree\include\opencv2\nonfree 路径下的 features2d.hpp 头文件中，可以发现这样两句定义：

```
typedef SURF SurfFeatureDetector;
typedef SURF SurfDescriptorExtractor;
```

我们都知道，`typedef` 声明是为现有类型创建一个新的名字，类型别名。这就表示，SURF 类忽然同时有了两个新名字 SurfFeatureDetector 和 SurfDescriptorExtractor。

也就是说，我们平常使用的 SurfFeatureDetector 类和 SurfDescriptorExtractor 类，其实就是 SURF 类，它们三者等价。

然后在这两句定义的上方，可以看到 SURF 类的类声明全貌，在这里由于篇幅原因，只贴出其轮廓。

```
class CV_EXPORTS_W SURF : public Feature2D
{
//.....
};
```

可以观察到，SURF 类公共继承自 Feature2D 类，我们再次进行转到定义，可以在路径…\opencv\build\include\opencv2\features2d\features2d.hpp 看到 Feature2D 类的声明。

```
class CV_EXPORTS_W Feature2D : public FeatureDetector, public
DescriptorExtractor
{
//.....
};
```

显然，Feature2D 类又是公共继承自 FeatureDetector 以及 DescriptorExtractor 类。继续刨根问底，看看其父类 FeatureDetector 以及 DescriptorExtractor 类的定义。

首先是 FeatureDetector 类。

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
class CV_EXPORTS_W FeatureDetector : public virtual Algorithm
{
//.....
};
```

这里，我们看到了以后经常会用到的 `detect()` 方法重载的两个原型，原来是 SURF 类经过两层的继承，从 `FeatureDetector` 类继承而来的。

```
CV_WRAP void detect(const Mat& image, CV_OUT vector<KeyPoint>&
keypoints, const Mat& mask=Mat()) const;

void detect(const vector<Mat>& images, vector<vector<KeyPoint> >&
keypoints, const vector<Mat>& masks=vector<Mat>()) const;
```

同样，看看 SURF 类的另一个“爷爷” `DescriptorExtractor` 类的声明。

```
class CV_EXPORTS_W DescriptorExtractor : public virtual Algorithm
{
//.....
};
```

上述代码表明 `FeatureDetector` 类和 `DescriptorExtractor` 类都虚继承自 `Algorithm` 基类。

终于，我们找到 SURF 类“德高望重”的祖先——OpenCV 中的 `Algorithm` 基类。其原型声明如下。

```
class CV_EXPORTS_W Algorithm
{
//.....
};
```

关于这几个类看似错综复杂的关系，将其绘制出来便一目了然了，如图 11.7 所示。

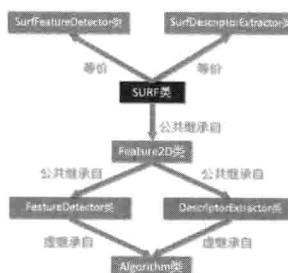


图 11.7 SURF 相关类关系图示

#### 11.1.4 绘制关键点：`drawKeypoints()` 函数

因为接下来的示例程序需要用到 `drawKeypoints` 函数，在这里顺便讲一讲。顾名思义，此函数用于绘制关键点。

```
C++: void drawKeypoints(const Mat&image, const vector<KeyPoint>&
keypoints, Mat& outImage, const Scalar& color=Scalar::all(-1), int
flags=DrawMatchesFlags::DEFAULT )
```

- 第一个参数，`const Mat&`类型的 `src`，输入图像。
- 第二个参数，`const vector<KeyPoint>&`类型的 `keypoints`，根据源图像得到的特征点。它是一个输出参数。
- 第三个参数，`Mat&`类型的 `outImage`，输出图像，其内容取决于第五个参数标识符 `flags`。
- 第四个参数，`const Scalar&`类型的 `color`，关键点的颜色，有默认值 `Scalar::all(-1)`。
- 第五个参数，`int`类型的 `flags`，绘制关键点的特征标识符，有默认值 `DrawMatchesFlags::DEFAULT`。可以在下面这个结构体中选取值。

```
struct DrawMatchesFlags
{
    enum
    {
        DEFAULT = 0, // 创建输出图像矩阵(使用 Mat::create)。使用现存的输出图像
        DRAW_OVER_OUTIMG = 1, //不创建输出图像矩阵，而是在输出图像上绘制匹配对
        NOT_DRAW_SINGLE_POINTS = 2, //单点特征点不被绘制
        DRAW_RICH_KEYPOINTS = 4 //对每一个关键点，绘制带大小和方向的关键点圆圈
    };
};
```

### 11.1.5 KeyPoint 类

`KeyPoint` 类是一个为特征点检测而生的数据结构，用于表示特征点。

```
class KeyPoint
{
    Point2f pt; //坐标
    float size; //特征点邻域直径
    float angle; //特征点的方向，值为[零,三百六十)，负值表示不使用
    float response;
    int octave; //特征点所在的图像金字塔的组
    int class_id; //用于聚类的 id
}
```

### 11.1.6 示例程序：SURF 特征点检测

这个示例程涉及到如下三个方面：

- 使用 `FeatureDetector` 接口来发现感兴趣点。
- 使用 `SurfFeatureDetector` 以及其函数 `detect` 来实现检测过程
- 使用函数 `drawKeypoints` 绘制检测到的关键点。

详细注释的源代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----#
#include "opencv2/core/core.hpp"
```

```
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/nonfree.hpp"
#include <iostream>
using namespace cv;

//-----【 main() 函数 】-----
// 描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----
int main()
{
    //【 0 】改变 console 字体颜色
    system("color 2F");

    //【 0 】显示帮助文字
    ShowHelpText();

    //【 1 】载入源图片并显示
    Mat srcImage1 = imread("1.jpg", 1 );
    Mat srcImage2 = imread("2.jpg", 1 );
    if( !srcImage1.data || !srcImage2.data )//检测是否读取成功
    { printf("读取图片错误，请确定目录下是否有 imread 函数指定名称的图片存在~!\n");
        return false; }
    imshow("原始图 1",srcImage1);
    imshow("原始图 2",srcImage2);

    //【 2 】定义需要用到的变量和类
    int minHessian = 400;//定义 SURF 中的 hessian 阈值特征点检测算子
    SurfFeatureDetector detector( minHessian );//定义一个
    SurfFeatureDetector( SURF ) 特征检测类对象
    std::vector<KeyPoint> keypoints_1, keypoints_2;//vector 模板类是能够
    存放任意类型的动态数组，能够增加和压缩数据

    //【 3 】调用 detect 函数检测出 SURF 特征关键点，保存在 vector 容器中
    detector.detect( srcImage1, keypoints_1 );
    detector.detect( srcImage2, keypoints_2 );

    //【 4 】绘制特征关键点
    Mat img_keypoints_1; Mat img_keypoints_2;
    drawKeypoints( srcImage1, keypoints_1, img_keypoints_1,
    Scalar::all(-1), DrawMatchesFlags::DEFAULT );
    drawKeypoints( srcImage2, keypoints_2, img_keypoints_2,
    Scalar::all(-1), DrawMatchesFlags::DEFAULT );

    //【 5 】显示效果图
    imshow("特征点检测效果图 1", img_keypoints_1 );
    imshow("特征点检测效果图 2", img_keypoints_2 );

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

示例程序截图如图 11.8~11.13 所示。

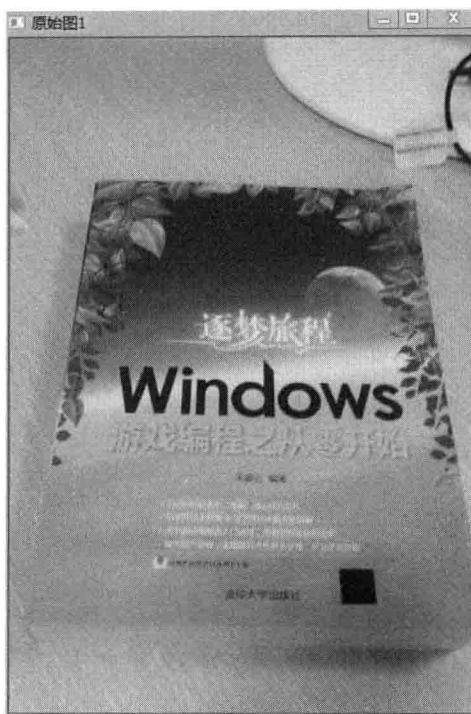


图 11.8 原始图 (1)

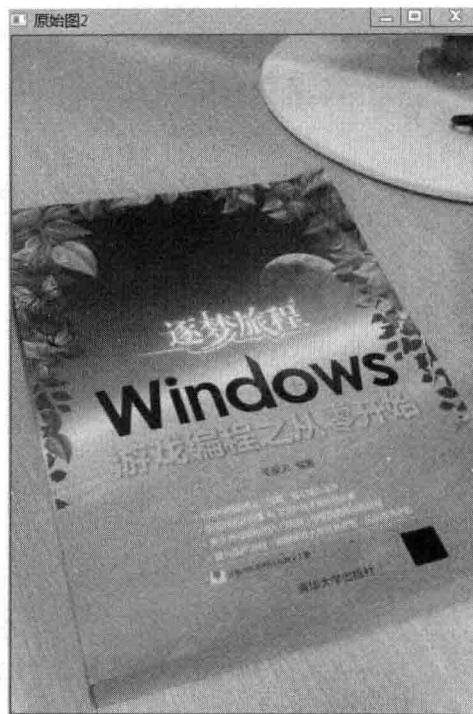


图 11.9 原始图 (2)

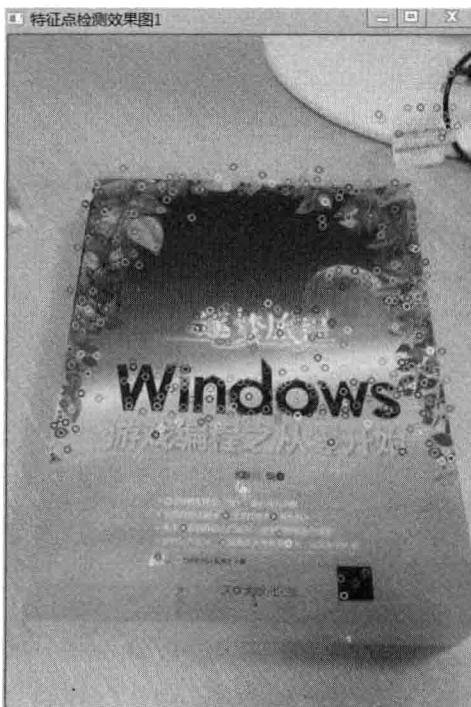


图 11.10 特征检测效果图 1  
(DEFAULT 绘制模式)

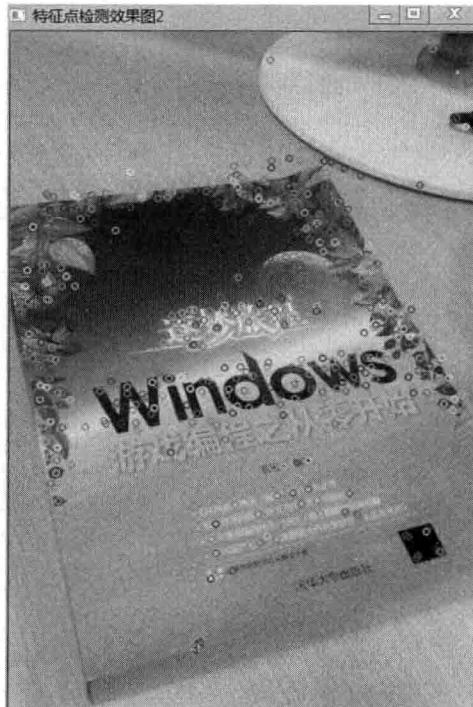


图 11.11 特征检测效果图 2  
(DEFAULT 绘制模式)

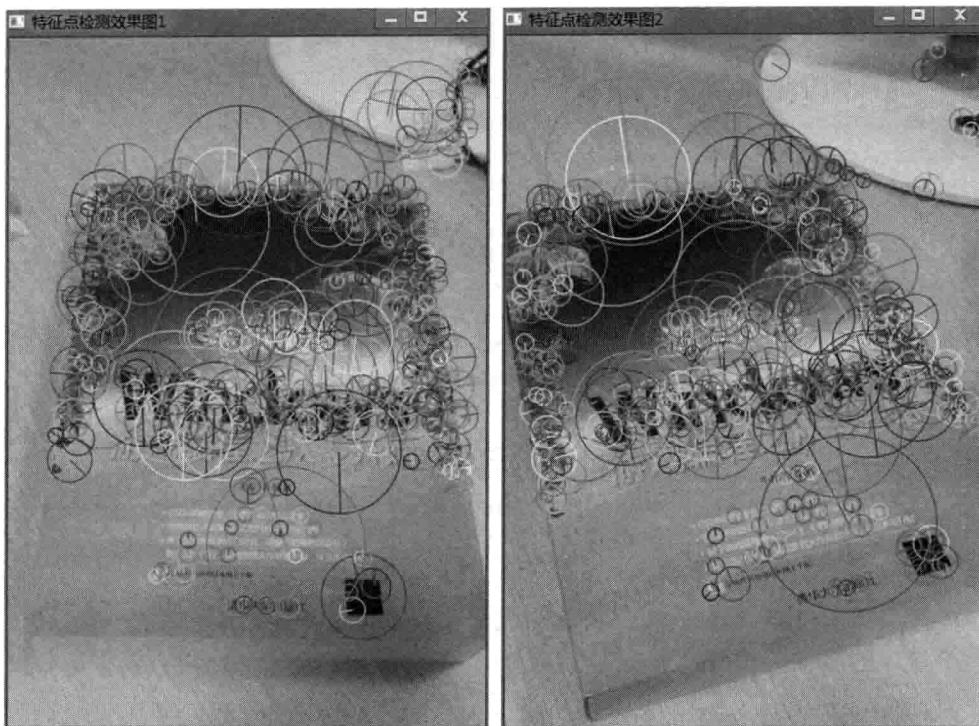


图 11.12 特征检测效果图 1  
(DRAW\_RICH\_KEYPOINTS 绘制模式)

图 11.13 特征检测效果图 2  
(DRAW\_RICH\_KEYPOINTS 绘制模式)

## 11.2 SURF 特征提取

通过上一节 SURF 相关内容的讲解，我们已经对 SURF 算法有了一定的理解。SURF 算法为每个检测到的特征定义了位置和尺度，尺度值可用于定义围绕特征点的窗口大小。不论物体的尺度在窗口是什么样的，都将包含相同的视觉信息，这些信息用于表示特征点以使得它们与众不同。

在特征匹配中，特征描述子通常用于  $N$  维向量，在光照不变以及少许透视变形的情况下很理想。另外，优质的描述子可以通过简单的距离测量进行比较，比如欧氏距离。因此，它们在特征匹配算法中，用处是很大的。

在 OpenCV 中，使用 SURF 进行特征点描述主要是 `drawMatches` 方法和 `BruteForceMatcher` 类的运用，下面让我们一起来认识它们。

### 11.2.1 绘制匹配点：`drawMatches()` 函数

`drawMatches` 用于绘制出相匹配的两个图像的关键点，它有以下两个版本的 C++ 函数原型。

```
C++: void drawMatches(const Mat& img1,  
const vector<KeyPoint>& keypoints1,  
const Mat& img2,  
const vector<KeyPoint>& keypoints2,
```

```

const vector<DMatch>& matches1to2,
Mat& outImg,
const Scalar& matchColor=Scalar::all(-1),
const Scalar& singlePointColor=Scalar::all(-1),
const vector<char>& matchesMask=vector<char>(),
int flags=DrawMatchesFlags::DEFAULT )

C++: void drawMatches(const Mat& img1,
const vector<KeyPoint>& keypoints1,
const Mat& img2,
const vector<KeyPoint>& keypoints2,
const vector<vector<DMatch>>& matches1to2,
Mat& outImg,
const Scalar& matchColor=Scalar::all(-1),
const Scalar& singlePointColor=Scalar::all(-1),
const vector<vector<char>>& matchesMask=vector<vector<char>>(),
int flags=DrawMatchesFlags::DEFAULT )

```

除了第五个参数 matches1to2 和第九个参数 matchesMask 有细微的差别以外，两个版本的基本上相同。下面统一讲解。

- 第一个参数，const Mat&类型的 img1，第一幅源图像。
- 第二个参数，const vector<KeyPoint>&类型的 keypoints1，根据第一幅源图像得到的特征点，它是一个输出参数。
- 第三个参数，const Mat&类型的 img2，第二幅源图像。
- 第四个参数，const vector<KeyPoint>&类型的 keypoints2，根据第二幅源图像得到的特征点，它是一个输出参数。
- 第五个参数，matches1to2，第一幅图像到第二幅图像的匹配点，即表示每一个图 1 中的特征点都在图 2 中有一一对应的点、
- 第六个参数，Mat&类型的 outImg，输出图像，其内容取决于第五个参数标识符 flags。
- 第七个参数，const Scalar&类型的 matchColor，匹配的输出颜色，即线和关键点的颜色。它有默认值 Scalar::all(-1)，表示颜色是随机生成的。
- 第八个参数，const Scalar&类型的 singlePointColor，单一特征点的颜色，它也表示随机生成颜色的默认值 Scalar::all(-1)。
- 第九个参数，matchesMask，确定哪些匹配是会绘制出来的掩膜，如果掩膜为空，表示所有匹配都进行绘制。
- 第十个参数，int 类型的 flags，特征绘制的标识符，有默认值 DrawMatchesFlags::DEFAULT。可以在如下这个 DrawMatchesFlags 结构体中选取值：

```

struct DrawMatchesFlags
{
    enum
    {
        DEFAULT = 0, // 创建输出图像矩阵(使用 Mat::create)。使用现存的输出图像

```

```

绘制匹配对和特征点。且对每一个关键点，只绘制中间点
DRAW_OVER_OUTIMG = 1, //不创建输出图像矩阵，而是在输出图像上绘制匹配对
NOT_DRAW_SINGLE_POINTS = 2, //单点特征点不被绘制
DRAW_RICH_KEYPOINTS = 4 //对每一个关键点，绘制带大小和方向的关键点圆圈
};

};

```

## 11.2.2 BruteForceMatcher 类源码分析

接下来，我们看看本节中会用到的 BruteForceMatcher 类的源码分析，BruteForceMatcher 类用于进行暴力匹配相关的操作。在……\opencv\sources\modules\legacy\include\opencv2\legacy\legacy.hpp 路径下，可以找到 BruteForceMatcher 类的定义。

```

template<class Distance>
class CV_EXPORTS BruteForceMatcher : publicBFMatcher
{
public:
    BruteForceMatcher( Distance d = Distance() ) :
        BFMatcher(Distance::normType, false) { (void)d; }
    virtual ~BruteForceMatcher() {}
};


```

其公共继承自 BFMatcher 类。而 BFMatcher 类位于…\opencv\sources\modules\features2d\include\opencv2\features2d\features2d.hpp 路径之下，我们一起看看其代码。

```

class CV_EXPORTS_W BFMatcher : publicDescriptorMatcher
{
//.....
};


```

发现公共其继承自 DescriptorMatcher 类，再次进行溯源，在 D:\Program Files(x86)\opencv\sources\modules\features2d\include\opencv2\features2d\features2d.hpp 路径下找到 DescriptorMatcher 类的定义。

```

class CV_EXPORTS_W DescriptorMatcher :public Algorithm
{
//.....
};


```

可以发现，DescriptorMatcher 类和之前我们讲到 FeatureDetector 类和 DescriptorExtractor 类一样，都是继承自它们“德高望重的祖先”Algorithm 基类的。

而用 BruteForceMatcher 类时用到最多的 match 方法，是它从 DescriptorMatcher 类那里继承而来。定义如下。

```

//为各种描述符找到一个最佳的匹配（若掩膜为空）
CV_WRAP void match( const Mat& queryDescriptors, const
Mat&trainDescriptors,
                    CV_OUTvector<DMatch>& matches, const Mat& mask=Mat() ) const;

```

### 11.2.3 示例程序：SURF 特征提取

这个示例程序中，我们利用 `SurfDescriptorExtractor` 类进行特征向量的相关计算。程序利用了 SURF 特征的特征描述办法，其操作封装在类 `SurfFeatureDetector` 中，利用类内的 `detect` 函数可以检测出 SURF 特征的关键点，保存在 `vector` 容器中。第二步利用 `SurfDescriptorExtractor` 类进行特征向量的相关计算。将之前的 `vector` 变量变成向量矩阵形式保存在 `Mat` 中。最后强行匹配两幅图像的特征向量，利用了类 `BruteForceMatcher` 中的函数 `match`。

程序的核心思想如下。

- 使用 `DescriptorExtractor` 接口来寻找关键点对应的特征向量。
- 使用 `SurfDescriptorExtractor` 以及它的函数 `compute` 来完成特定的计算。
- 使用 `BruteForceMatcher` 来匹配特征向量。
- 使用函数 `drawMatches` 来绘制检测到的匹配点。

关键点讲解：OpenCV2 引入了一个通用类，用于提取不同的特征点描述子，计算如下。

```
//【4】计算描述子（特征向量）
SurfDescriptorExtractor extractor;
Mat descriptors1, descriptors2;
extractor.compute(srcImage1, keyPoint1, descriptors1 );
extractor.compute(srcImage2, keyPoints2, descriptors2 );
```

这里的结果为一个 `Mat` 矩阵，它的行数与特征点向量中元素个数是一致的。每行都是一个  $N$  维描述子的向量，比如 SURF 算法默认的描述子维度为 64，该向量描绘了特征点周围的强度样式。两个特征点越相似，它们的特征向量也越靠近。这些描述子在图像匹配中尤其有用。例如，我们想匹配同一个场景中的两幅图像。首先，检测每幅图像中的特征，然后提取它们的描述子。第一幅图像中的每一个特征描述子向量都会与第二幅图中的描述子进行比较，得分最高的一对描述子（也就是两个向量的距离最近）将被视为那个特征的最佳匹配。该过程对于第一幅图像中的所有特征进行重复，这便是 `BruteForceMatcher` 中实行的最基本的策略。相关代码如下。

```
//【5】使用 BruteForce 进行匹配
//实例化一个匹配器
BruteForceMatcher<L2<float>> matcher;
std::vector<DMatch> matches;
//匹配两幅图中的描述子（descriptors）
matcher.match(descriptors1, descriptors2, matches );
```

`BruteForceMatcher` 是由 `DescriptorMatcher` 派生出来的一个类，而 `DescriptorMatcher` 定义了不同的匹配策略的共同接口。调用 `match` 方法后，在其第三个参数输出一个 `cv::DMatch` 向量。于是我们定义一个 `std::vector<DMatch>` 类型的 `matches`。

调用 `match` 方法之后，便可以使用 `drawMatches` 方法对匹配到的点进行绘制，

并最终显示出来。相关代码如下。

```
//【6】绘制从两个图像中匹配出的关键点
Mat imgMatches;
drawMatches(srcImage1, keyPoint1, srcImage2, keyPoints2, matches,
imgMatches );//进行绘制
//【7】显示效果图
imshow("匹配图", imgMatches );
```

核心部分讲解完毕，下面我们一起来看看详细注释的示例程序的完全体。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----  

#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <opencv2/nonfree/nonfree.hpp>
#include<opencv2/legacy/legacy.hpp>
#include <iostream>
using namespace cv;
using namespace std;

-----【main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----  

int main()
{
    //【1】载入素材图
    Mat srcImage1 = imread("1.jpg",1);
    Mat srcImage2 = imread("2.jpg",1);
    if( !srcImage1.data || !srcImage2.data )
    { printf("读取图片错误,请确定目录下是否有 imread 函数指定的图片存在~! \n");
    return false; }

    //【2】使用 SURF 算子检测关键点
    int minHessian = 700;//SURF 算法中的 hessian 阈值
    SurfFeatureDetector detector( minHessian );//定义一个
SurfFeatureDetector ( SURF ) 特征检测类对象
    std::vector<KeyPoint> keyPoint1, keyPoints2;//vector 模板类，存放任意
类型的动态数组

    //【3】调用 detect 函数检测出 SURF 特征关键点，保存在 vector 容器中
    detector.detect( srcImage1, keyPoint1 );
    detector.detect( srcImage2, keyPoints2 );

    //【4】计算描述符（特征向量）
    SurfDescriptorExtractor extractor;
    Mat descriptors1, descriptors2;
    extractor.compute( srcImage1, keyPoint1, descriptors1 );
    extractor.compute( srcImage2, keyPoints2, descriptors2 );

    //【5】使用 BruteForce 进行匹配
```

```
// 实例化一个匹配器  
BruteForceMatcher< L2<float> > matcher;  
std::vector< DMatch > matches;  
// 匹配两幅图中的描述子 (descriptors)  
matcher.match( descriptors1, descriptors2, matches );  
  
// 【6】绘制从两个图像中匹配出的关键点  
Mat imgMatches;  
drawMatches(srcImage1, keyPoint1, srcImage2, keyPoints2, matches,  
imgMatches); // 进行绘制  
  
// 【7】显示效果图  
imshow("匹配图", imgMatches );  
  
waitKey(0);  
return 0;  
}
```

让我们看看运行效果图（图 11.14）。

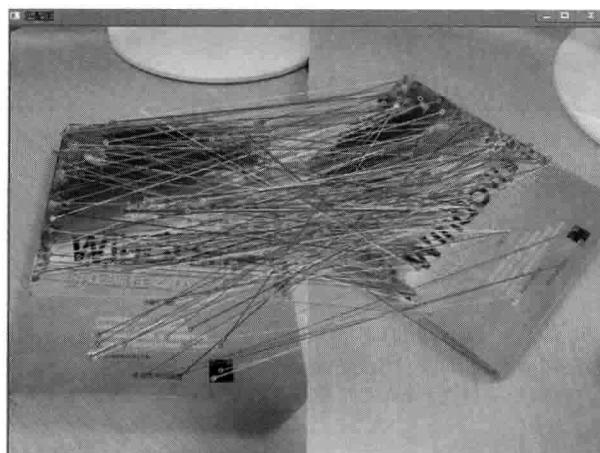


图 11.14 SURF 特征描述效果图

## 11.3 使用 FLANN 进行特征点匹配

本节，我们将学习如何使用 FlannBasedMatcher 接口以及函数 FLANN()，实现快速高效匹配（快速最近邻逼近搜索函数库，Fast Library for Approximate Nearest Neighbors，FLANN）。

### 11.3.1 FlannBasedMatcher 类的简单分析

在 OpenCV 源代码中，找到 FlannBasedMatcher 类的脉络如下。

```
class CV_EXPORTS_W FlannBasedMatcher : public DescriptorMatcher  
{  
//.....  
};
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

可以发现 FlannBasedMatcher 类也是继承自 DescriptorMatcher，并且同样主要使用来自 DescriptorMatcher 类的 match 方法进行匹配。下面，让我们讲解一下此类方法的用法。

### 11.3.2 找到最佳匹配：DescriptorMatcher::match 方法

DescriptorMatcher::match()函数从每个描述符查询集中找到最佳匹配，有两个版本的源码，下面用注释对其进行讲解。

```
C++: void DescriptorMatcher::match(
    const Mat& queryDescriptors, //查询描述符集
    const Mat& trainDescriptors, //训练描述符集
    vector<DMatch>& matches, //得到的匹配。若查询描述符有在掩膜中被标记出来，则没有匹配添加到描述符中。则匹配量可能会比查询描述符数量少
    const Mat& mask=Mat() ) //指定输入查询和训练描述符允许匹配的掩膜

C++: void DescriptorMatcher::match(
    const Mat& queryDescriptors, //查询描述符集
    vector<DMatch>& matches, //得到的匹配。若查询描述符有在掩膜中被标记出来，则没有匹配添加到描述符中，则匹配量可能会比查询描述符数量少
    const vector<Mat>& masks=vector<Mat>() ) //一组掩膜，每个 masks[i] 从第 i 个图像 trainDescCollection[i] 指定输入查询和训练描述符允许匹配的掩膜
```

### 11.3.3 示例程序：使用 FLANN 进行特征点匹配

本节我们将演示如何使用 FLANN 进行特征点匹配，详细注释的示例程序代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <opencv2/nonfree/nonfree.hpp>
#include<opencv2/legacy/legacy.hpp>
#include <iostream>
using namespace cv;
using namespace std;

-----【main() 函数】-----
//  描述：控制台应用程序的入口函数，我们的程序从这里开始执行
-----
int main( int argc, char** argv )
{
    //【1】载入源图片
    Mat img_1 = imread("1.jpg", 1 );
    Mat img_2 = imread( "2.jpg", 1 );
    if( !img_1.data || !img_2.data ) { printf("读取图片 image0 错误~! \n"); return false; }
```

```
//【2】利用 SURF 检测器检测的关键点
int minHessian = 300;
SURF detector( minHessian );
std::vector<KeyPoint> keypoints_1, keypoints_2;
detector.detect( img_1, keypoints_1 );
detector.detect( img_2, keypoints_2 );

//【3】计算描述符（特征向量）
SURF extractor;
Mat descriptors_1, descriptors_2;
extractor.compute( img_1, keypoints_1, descriptors_1 );
extractor.compute( img_2, keypoints_2, descriptors_2 );

//【4】采用 FLANN 算法匹配描述符向量
FlannBasedMatcher matcher;
std::vector< DMatch > matches;
matcher.match( descriptors_1, descriptors_2, matches );
double max_dist = 0; double min_dist = 100;

//【5】快速计算关键点之间的最大和最小距离
for( int i = 0; i < descriptors_1.rows; i++ )
{
    double dist = matches[i].distance;
    if( dist < min_dist ) min_dist = dist;
    if( dist > max_dist ) max_dist = dist;
}
//输出距离信息
printf("> 最大距离 (Max dist) : %f \n", max_dist );
printf("> 最小距离 (Min dist) : %f \n", min_dist );

//【6】存下符合条件的匹配结果（即其距离小于 2* min_dist 的），使用 radiusMatch
同样可行
std::vector< DMatch > good_matches;
for( int i = 0; i < descriptors_1.rows; i++ )
{
    if( matches[i].distance < 2*min_dist )
        { good_matches.push_back( matches[i] ); }
}

//【7】绘制出符合条件的匹配点
Mat img_matches;
drawMatches( img_1, keypoints_1, img_2, keypoints_2,
    good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
    vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );

//【8】输出相关匹配点信息
for( int i = 0; i < good_matches.size(); i++ )
    { printf( ">符合条件的匹配点 [%d] 特征点 1: %d -- 特征点 2: %d \n", i,
    good_matches[i].queryIdx, good_matches[i].trainIdx ); }

//【9】显示效果图
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
imshow( "匹配效果图", img_matches );
//按任意键退出程序
waitKey(0);
return 0;
}
```

让我们一起看看程序的运行截图。如图 11.15、11.16。

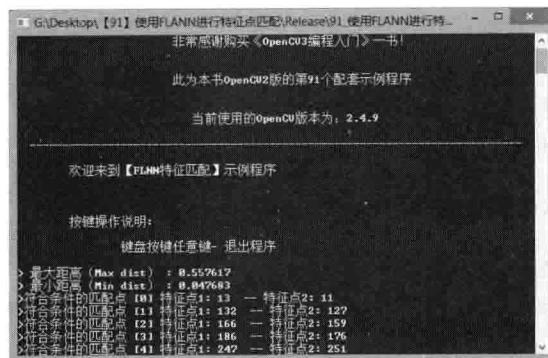


图 11.15 匹配信息窗口



图 11.16 匹配效果图

#### 11.3.4 综合示例程序：FLANN 结合 SURF 进行关键点的描述和匹配

通过上文的学习，我们已经了解了 FLANN 算法和 SURF 算法。

而这个综合示例程序中，将两者结合起来使用——用 SURF 进行关键点和描述符的提取，用 FLANN 进行匹配。

知识点和讲解点都蕴含在程序代码中，不妨整体贴出经过详细注释的程序代码，方便大家把握程序脉络。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
//-----  

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
```

[Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com](http://www.simpopdf.com)

```
#include <opencv2/nonfree/features2d.hpp>
#include <opencv2/features2d/features2d.hpp>
using namespace cv;
using namespace std;

//-----【 main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----
int main()
{
    //【0】改变 console 字体颜色
    system("color 6F");

    //【1】载入图像、显示并转化为灰度图
    Mat trainImage = imread("1.jpg"), trainImage_gray;
    imshow("原始图",trainImage);
    cvtColor(trainImage, trainImage_gray, CV_BGR2GRAY);

    //【2】检测 Surf 关键点、提取训练图像描述符
    vector<KeyPoint> train_keyPoint;
    Mat trainDescriptor;
    SurfFeatureDetector featureDetector(80);
    featureDetector.detect(trainImage_gray, train_keyPoint);
    SurfDescriptorExtractor featureExtractor;
    featureExtractor.compute(trainImage_gray, train_keyPoint,
    trainDescriptor);

    //【3】创建基于 FLANN 的描述符匹配对象
    FlannBasedMatcher matcher;
    vector<Mat> train_desc_collection(1, trainDescriptor);
    matcher.add(train_desc_collection);
    matcher.train();

    //【4】创建视频对象、定义帧率
    VideoCapture cap(0);
    unsigned int frameCount = 0;//帧数

    //【5】不断循环，直到 q 键被按下
    while(char(waitKey(1)) != 'q')
    {
        //<1>参数设置
        int64 time0 = getTickCount();
        Mat testImage, testImage_gray;
        cap >> testImage;//采集视频到 testImage 中
        if(testImage.empty())
            continue;

        //<2>转化图像到灰度
        cvtColor(testImage, testImage_gray, CV_BGR2GRAY);

        //<3>检测 S 关键点、提取测试图像描述符
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
vector<KeyPoint> test_keyPoint;
Mat testDescriptor;
featureDetector.detect(testImage_gray, test_keyPoint);
featureExtractor.compute(testImage_gray, test_keyPoint,
testDescriptor);

//<4>匹配训练和测试描述符
vector<vector<DMatch>> matches;
matcher.knnMatch(testDescriptor, matches, 2);

// <5>根据劳氏算法 (Lowe's algorithm), 得到优秀的匹配点
vector<DMatch> goodMatches;
for(unsigned int i = 0; i < matches.size(); i++)
{
    if(matches[i][0].distance < 0.6 * matches[i][1].distance)
        goodMatches.push_back(matches[i][0]);
}

//<6>绘制匹配点并显示窗口
Mat dstImage;
drawMatches(testImage, test_keyPoint, trainImage,
train_keyPoint, goodMatches, dstImage);
imshow("匹配窗口", dstImage);

//<7>输出帧率信息
cout << "当前帧率为: " << getTickFrequency() / (getTickCount() -
time0) << endl;
}

return 0;
}
```

程序代码就是如上这些，接着，我们一起看看程序运行截图。如图 11.17、11.18 所示。



图 11.17 匹配原始图

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

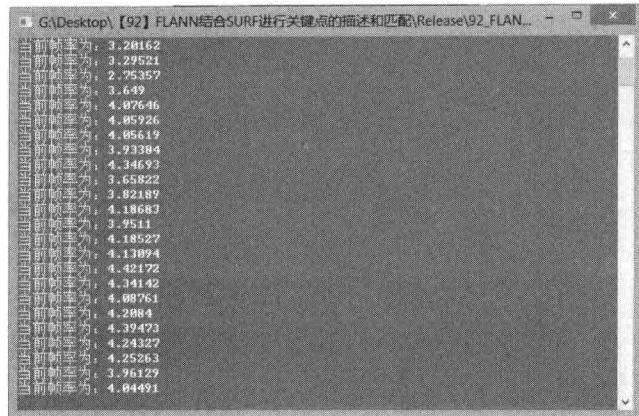


图 11.18 输出帧率的窗口

可以发现，用此方法进行匹配时帧率偏低，算法效率有待加强。

当找一本和原图不相关的书籍对准摄像头，得到的匹配点寥寥无几。如图 11.19 所示。

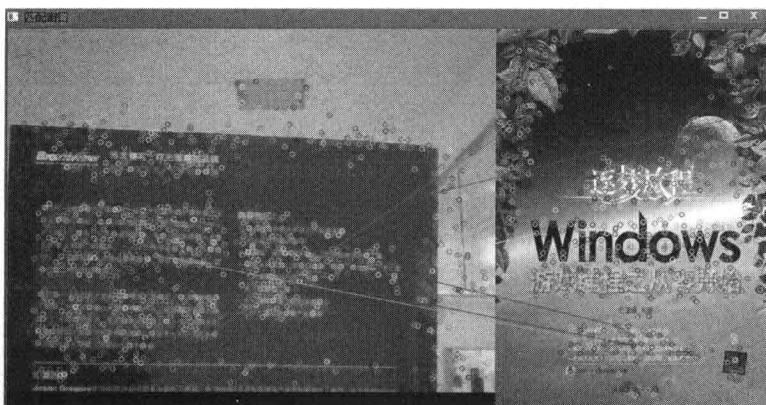


图 11.19 匹配点寥寥无几

而当找到和原始图对应的书本封面面对准摄像头时，得到的匹配点多且整齐。如图 11.20 所示。

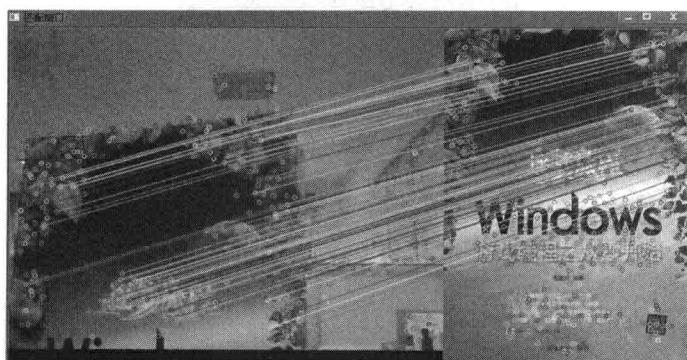


图 11.20 整齐的匹配点

### 11.3.5 综合示例程序：SIFT 配合暴力匹配进行关键点描述和提取

在上文刚刚讲过 SURF 算法相关的内容，而 SURF 算法作为 SIFT 算法的加速版，理应当有更快的检测速度。表 11.1 是两者一个简单的比较。

表 11.1 SURF 算法-SIFT 算法各项指标对比

比较内容	SIFT	SURF
尺度空间	DOG 与不同尺度的图片卷积	不同尺度的 box filters 与原图片卷积
特征点检测	先进行非极大抑制，再去除低对比度的点。再通过 Hessian 矩阵去除边缘的点	先利用 Hessian 矩阵确定候选点，然后进行非极大抑制
方向	在正方形区域内统计梯度的幅值的直方图，找 max 对应的方向。可以有多个方向。	在圆形区域内，计算各个扇形范围内 x、y 方向的 haar 小波响应，找模最大的扇形方向
特征描述子	16*16 的采样点划分为 4*4 的区域，计算每个区域的采样点的梯度方向和幅值，统计成 8bin 直方图，一共 4*4*8=128 维	20*20s 的区域划分为 4*4 的子区域，每个子区域找 5*5 个采样点，计算采样点的 haar 小波响应，记录 $\sum dx$ , $\sum dy$ , $\sum  dx $ , $\sum  dy $ , 一共 4*4*4=64 维

且理论上，SURF 是 SIFT 速度的 3 倍。

本节我们来看一个用 SIFT 算法配合暴力匹配的示例程序，看看是否 SURF 有比 SIFT 更快的匹配速度（由于匹配方式不同，这里只是大概的比较）。

可以发现，其代码形式和上一节中的代码基本类似。

```

//-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----[main() 函数]-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----[1]载入图像、显示并转化为灰度图
int main()
{
    //【0】改变 console 字体颜色
    system("color 5F");

    //【1】载入图像、显示并转化为灰度图
    Mat trainImage = imread("l.jpg"), trainImage_gray;

```

```
imshow("原始图",trainImage);
cvtColor(trainImage, trainImage_gray, CV_BGR2GRAY);

//【2】检测 SIFT 关键点、提取训练图像描述符
vector<KeyPoint> train_keyPoint;
Mat trainDescription;
SiftFeatureDetector featureDetector;
featureDetector.detect(trainImage_gray, train_keyPoint);
SiftDescriptorExtractor featureExtractor;
featureExtractor.compute(trainImage_gray, train_keyPoint,
trainDescription);

// 【3】进行基于描述符的暴力匹配
BFMatcher matcher;
vector<Mat> train_desc_collection(1, trainDescription);
matcher.add(train_desc_collection);
matcher.train();

//【4】创建视频对象、定义帧率
VideoCapture cap(0);
unsigned int frameCount = 0;//帧数

//【5】不断循环，直到 q 键被按下
while(char(waitKey(1)) != 'q')
{
    //<1>参数设置
    double time0 = getTickCount();
    Mat captureImage, captureImage_gray;
    cap >> captureImage;//采集视频到 testImage 中
    if(captureImage.empty())
        continue;

    //<2>转化图像到灰度
    cvtColor(captureImage, captureImage_gray, CV_BGR2GRAY);

    //<3>检测 SURF 关键点、提取测试图像描述符
    vector<KeyPoint> test_keyPoint;
    Mat testDescriptor;
    featureDetector.detect(captureImage_gray, test_keyPoint);
    featureExtractor.compute(captureImage_gray, test_keyPoint,
testDescriptor);

    //<4>匹配训练和测试描述符
    vector<vector<DMatch> > matches;
    matcher.knnMatch(testDescriptor, matches, 2);

    // <5>根据劳氏算法 (Lowe's algorithm)，得到优秀的匹配点
    vector<DMatch> goodMatches;
    for(unsigned int i = 0; i < matches.size(); i++)
    {
        if(matches[i][0].distance < 0.6 * matches[i][1].distance)
            goodMatches.push_back(matches[i][0]);
    }
}
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
}

//<6>绘制匹配点并显示窗口
Mat dstImage;
drawMatches(captureImage, test_keyPoint, trainImage,
train_keyPoint, goodMatches, dstImage);
imshow("匹配窗口", dstImage);

//<7>输出帧率信息
cout << "当前帧率为: " << getTickFrequency() / (getTickCount() -
time0) << endl;
}

return 0;
}
```

让我们一起看看运行截图。如图 11.21、11.22 所示。



图 11.21 原始图窗口

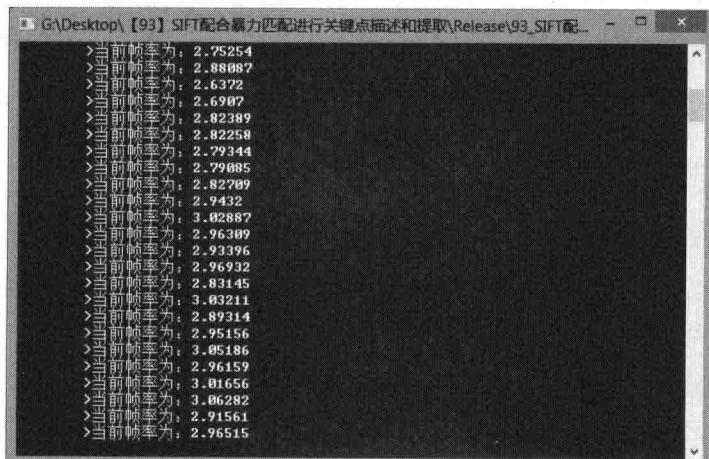


图 11.22 输出帧率的窗口

可以发现，在代码写法基本相同的情况下，用此方法进行匹配时帧率低于 SURF 算法的平均约 4.0 的帧率，算法效率有待加强。

当找一本和原图不相关的书籍对准摄像头，得到的匹配点寥寥无几。如图 11.23 所示。

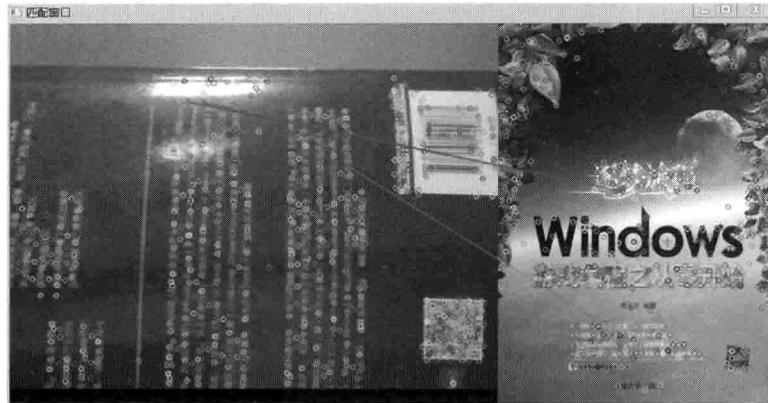


图 11.23 匹配点寥寥无几

而当找到和原始图对应的书对准摄像头时，得到的匹配点多且整齐。如图 11.24 所示。

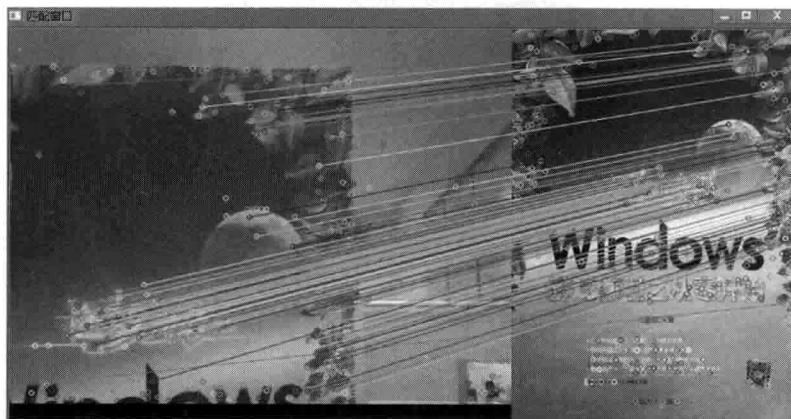


图 11.24 整齐的匹配点（暴力匹配）

## 11.4 寻找已知物体

在 FLANN 特征匹配的基础上，还可以进一步利用 Homography 映射找出已知物体。具体来说就是利用 `findHomography` 函数通过匹配的关键点找出相应的变换，再利用 `perspectiveTransform` 函数映射点群。

也就是分为以下两个大的步骤。

- (1) 使用函数 `findHomography` 寻找匹配上的关键点的变换。
- (2) 使用函数 `perspectiveTransform` 来映射点。

下面，我们分别对相关该函数进行简单介绍。

### 11.4.1 寻找透视变换：findHomography()函数

此函数的作用是找到并返回源图像和目标图像之间的透视变换 H:

$$s_i x'_i y'_i \sim H x_i y_i$$

```
C++: Mat findHomography(
InputArray srcPoints,
InputArray dstPoints,
int method=0,
double ransacReprojThreshold=3,
OutputArray mask=noArray() )
```

- 第一个参数，InputArray 类型的 srcPoints，源平面上的对应点，可以是 CV\_32FC2 的矩阵类型或者 vector<Point2f>。
- 第二个参数，InputArray 类型的 dstPoints，目标平面上的对应点，可以是 CV\_32FC2 的矩阵类型或者 vector<Point2f>。
- 第三个参数，int 类型的 method，用于计算单应矩阵的方法。可选标识符为如表 11.2 所示。

表 11.2 计算单应矩阵可选标识符

标识符	含义
0	使用所有点的常规方法（默认值）
CV_RANSAC	基于 RANSAC 的鲁棒性方法（robustmethod）
CV_LMEDS	最小中值鲁棒性方法

- 第四个参数，double 类型的 ransacReprojThreshold，有默认值 3，处理点对为内围层时，允许重投影误差的最大值。这就是说，当  $\| \text{dstPoints}_i - \text{convertPointsHomogeneous}(\text{H} * \text{srcPoints}_i) \| > \text{ransacReprojThreshold}$  时，这里的点 i 被看做内围层。若 srcPoints 和 dstPoints 是以像素为单位的，那么此参数的取值范围一般在 1 到 10 之间。
- 第五个参数，OutputArray 类型的 mask，有默认值 noArray()，是一个可选的参数，通过上面讲到的鲁棒性方法( CV\_RANSAC 或者 CV\_LMEDS )设置输出掩码。需要注意的是，输入掩码值会被忽略掉。

### 11.4.2 进行透视矩阵变换：perspectiveTransform()函数

perspectiveTransform 函数的作用是进行向量透视矩阵变换。

```
C++: void perspectiveTransform(InputArray src, OutputArray dst,
InputArray m)
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可，且需为双通道或三通道浮点型图像，其中的每个元素是二维或三

维可被转换的矢量。

- 第二个参数，OutputArray 类型的 dst，函数调用后的运算结果存在这里，即这个参数用于存放函数调用后的输出结果，需和源图片有一样的尺寸和类型。
- 第三个参数，InputArray 类型的 m，变换矩阵，为 3x3 或者 4x4 浮点型矩阵。

### 11.4.3 示例程序：寻找已知物体

本节我们将放出用 OpenCV 寻找已知物体的详细注释的程序源代码。

```

//-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
//-----

#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/nonfree/nonfree.hpp"
#include <iostream>
using namespace cv;
using namespace std;

//-----【main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----

int main()
{
    //【1】载入原始图片
    Mat srcImage1 = imread( "1.jpg", 1 );
    Mat srcImage2 = imread( "2.jpg", 1 );
    if( !srcImage1.data || !srcImage2.data )
        { printf("读取图片错误，请确定目录下是否有 imread 函数指定的图片存在~!\n");
        return false; }

    //【2】使用 SURF 算子检测关键点
    int minHessian = 400;//SURF 算法中的 hessian 阈值
    SurfFeatureDetector detector( minHessian );//定义一个
    SurfFeatureDetector ( SURF ) 特征检测类对象
    vector<KeyPoint> keypoints_object, keypoints_scene;//vector 模板类，
    存放任意类型的动态数组

    //【3】调用 detect 函数检测出 SURF 特征关键点，保存在 vector 容器中
    detector.detect( srcImage1, keypoints_object );
    detector.detect( srcImage2, keypoints_scene );

    //【4】计算描述符（特征向量）
    SurfDescriptorExtractor extractor;
    Mat descriptors_object, descriptors_scene;
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
extractor.compute( srcImage1, keypoints_object,
descriptors_object );
extractor.compute( srcImage2, keypoints_scene, descriptors_scene );

//【5】使用 FLANN 匹配算子进行匹配
FlannBasedMatcher matcher;
vector< DMatch > matches;
matcher.match( descriptors_object, descriptors_scene, matches );
double max_dist = 0; double min_dist = 100;//最小距离和最大距离

//【6】计算出关键点之间距离的最大值和最小值
for( int i = 0; i < descriptors_object.rows; i++ )
{
    double dist = matches[i].distance;
    if( dist < min_dist ) min_dist = dist;
    if( dist > max_dist ) max_dist = dist;
}

printf(">Max dist 最大距离 : %f \n", max_dist );
printf(">Min dist 最小距离 : %f \n", min_dist );

//【7】存下匹配距离小于 3*min_dist 的点对
std::vector< DMatch > good_matches;
for( int i = 0; i < descriptors_object.rows; i++ )
{
    if( matches[i].distance < 3*min_dist )
    {
        good_matches.push_back( matches[i] );
    }
}

//绘制出匹配到的关键点
Mat img_matches;
drawMatches( srcImage1, keypoints_object, srcImage2,
keypoints_scene,
good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS );

//定义两个局部变量
vector<Point2f> obj;
vector<Point2f> scene;

//从匹配成功的匹配对中获取关键点
for( unsigned int i = 0; i < good_matches.size(); i++ )

{
    obj.push_back( keypoints_object[ good_matches[i].queryIdx ].pt );

    scene.push_back( keypoints_scene[ good_matches[i].trainIdx ].pt );
}

Mat H = findHomography( obj, scene, CV_RANSAC );//计算透视变换
```

```
//从待测图片中获取角点
vector<Point2f> obj_corners(4);
obj_corners[0] = cvPoint(0,0); obj_corners[1] =
cvPoint( srcImage1.cols, 0 );
obj_corners[2] = cvPoint( srcImage1.cols, srcImage1.rows );
obj_corners[3] = cvPoint( 0, srcImage1.rows );
vector<Point2f> scene_corners(4);

//进行透视变换
perspectiveTransform( obj_corners, scene_corners, H);

//绘制出角点之间的直线
line( img_matches, scene_corners[0] +
Point2f( static_cast<float>(srcImage1.cols), 0), scene_corners[1] +
Point2f( static_cast<float>(srcImage1.cols), 0), Scalar(255, 0, 123),
4 );
line( img_matches, scene_corners[1] +
Point2f( static_cast<float>(srcImage1.cols), 0), scene_corners[2] +
Point2f( static_cast<float>(srcImage1.cols), 0), Scalar( 255, 0, 123),
4 );
line( img_matches, scene_corners[2] +
Point2f( static_cast<float>(srcImage1.cols), 0), scene_corners[3] +
Point2f( static_cast<float>(srcImage1.cols), 0), Scalar( 255, 0, 123),
4 );
line( img_matches, scene_corners[3] +
Point2f( static_cast<float>(srcImage1.cols), 0), scene_corners[0] +
Point2f( static_cast<float>(srcImage1.cols), 0), Scalar( 255, 0, 123),
4 );

//显示最终结果
imshow( "Good Matches & Object detection", img_matches );

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

运行效果图如图 11.25、11.26 所示。

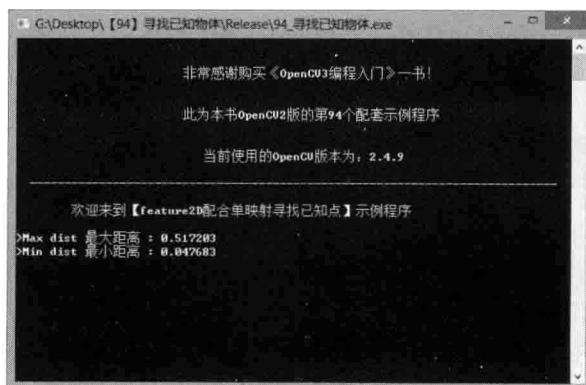


图 11.25 文本输出窗口

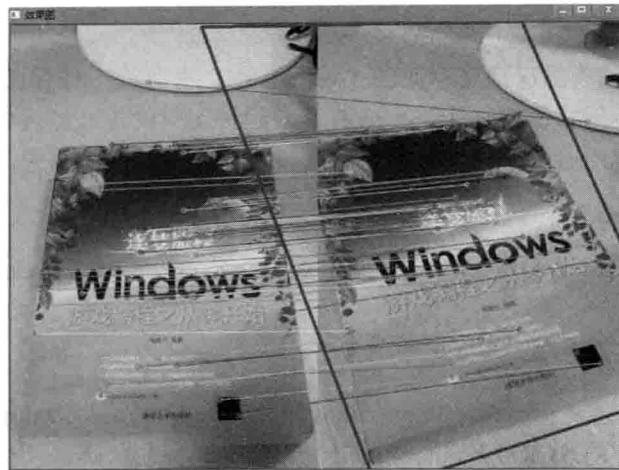


图 11.26 检测效果图

可以发现，图中用细线条匹配对应点，用粗线条指出检测到的物体。

## 11.5 ORB 特征提取

### 11.5.1 ORB 算法概述

ORB 为是 ORiented Brief 的简称，是 brief 算法的改进版。ORB 于 2011 年在《ORB: an efficient alternative to SIFT or SURF》这篇文章中被提出。此文章的摘要中说，ORB 算法比 sift 算法效率高两个数量级，而在计算速度上，ORB 是 sift 的 100 倍，是 surf 的 10 倍。而江湖上流传的说法是，ORB 算法综合性能在各种测评里相较于其他特征提取算法是最好的。

若想要引出 ORB (ORiented Brief) 算法，要由 Brief 描述子入手，下面，就来先介绍 Brief 描述子。

### 11.5.2 相关概念认知

#### 1. 关于 Brief 描述子

Brief 是 Binary Robust Independent Elementary Features 的缩写。这个特征描述子是由 EPFL 的 Calonder 在 ECCV2010 上提出的，主要思路就是在特征点附近随机选取若干点对，将这些点对的灰度值的大小，组合成一个二进制串，并将这个二进制串作为该特征点的特征描述子。

BRIEF 的优点在于速度，而缺点也相当明显。

- 不具备旋转不变性。
- 对噪声敏感
- 不具备尺度不变性。

而 ORB 算法就是试图解决上述缺点中的 1 和 2 提出的一种新概念。

## 2. 关于尺度不变性

ORB 没有试图解决尺度不变性（因为 FAST 本身就不具有尺度不变性），但是这样只求速度的特征描述子，一般都是应用在实时的视频处理中的，这样的话就可以通过跟踪还有一些启发式的策略来解决尺度不变性的问题。

## 3. 关于计算速度

经过统计，ORB 算法的执行速度是 SIFT 的 100 倍，是 SURF 的 10 倍。

### 11.5.3 ORB 类相关源码简单分析

同样是在路径 `...opencv\build\include\opencv2\features2d\features2d.hpp` 下，我们会发现和之前的 SURF 类源码分析时惊人相似的两句代码：

```
typedef ORB OrbFeatureDetector;
typedef ORB OrbDescriptorExtractor;
```

这就是说，ORB，OrbFeatureDetector，OrbDescriptorExtractor 这三个类，也是完全等价的。然后我们继续溯源，发现 ORB 类同样继承自 Feature2D 类：

```
class CV_EXPORTS_W ORB : public Feature2D
{
//.....
};
```

接下来的分析思路，和我们之前讲解 SURF 类的源码时一样，经过分析，可以得到如图 11.27 所示的类关系脉络图。

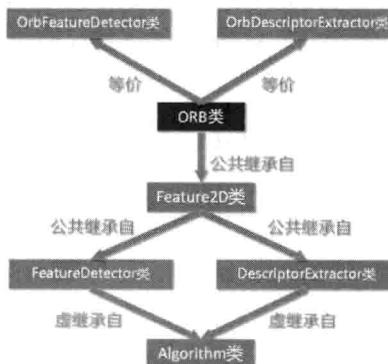


图 11.27 ORB 类相关联系脉络图

### 11.5.4 示例程序：ORB 算法描述与匹配

在与时俱进的 OpenCV 中，ORB 算法已被实现出来，而我们将其代码修改和注释，便得到了本期的示例程序。

此程序演示了 ORB 的关键点和描述符的提取，采用摄像头获取待检测图像，使用 FLANN-LSH 进行匹配。经过详细注释的代码如下。

```
//-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
-----  
#include <opencv2/opencv.hpp>  
#include <opencv2/highgui/highgui.hpp>  
#include <opencv2/nonfree/features2d.hpp>  
#include <opencv2/features2d/features2d.hpp>  
using namespace cv;  
using namespace std;  
  
-----【 main() 函数 】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行  
-----  
int main()  
{  
  
    //【 0 】载入源图，显示并转化为灰度图  
    Mat srcImage = imread("l.jpg");  
    imshow("原始图",srcImage);  
    Mat grayImage;  
    cvtColor(srcImage, grayImage, CV_BGR2GRAY);  
  
    //-----检测 SIFT 特征点并在图像中提取物体的描述符-----  
  
    //【 1 】参数定义  
    OrbFeatureDetector featureDetector;  
    vector<KeyPoint> keyPoints;  
    Mat descriptors;  
  
    //【 2 】调用 detect 函数检测出特征关键点，保存在 vector 容器中  
    featureDetector.detect(grayImage, keyPoints);  
  
    //【 3 】计算描述符（特征向量）  
    OrbDescriptorExtractor featureExtractor;  
    featureExtractor.compute(grayImage, keyPoints, descriptors);  
  
    //【 4 】基于 FLANN 的描述符对象匹配  
    flann::Index flannIndex(descriptors, flann::LshIndexParams(12, 20,  
2), cvflann::FLANN_DIST_HAMMING);  
  
    //【 5 】初始化视频采集对象  
    VideoCapture cap(0);  
    cap.set(CV_CAP_PROP_FRAME_WIDTH, 360); //设置采集视频的宽度  
    cap.set(CV_CAP_PROP_FRAME_HEIGHT, 900); //设置采集视频的高度  
  
    unsigned int frameCount = 0; //帧数  
  
    //【 6 】轮询，直到按下 ESC 键退出循环  
    while(1)  
    {  
        double time0 = static_cast<double>(getTickCount()); //记录起始时间  
        Mat captureImage, captureImage_gray; //定义两个 Mat 变量，用于视频  
        采集
```

[Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com](http://www.simpopdf.com)

```
cap >> captureImage;//采集视频帧
if( captureImage.empty())//采集为空的处理
    continue;

cvtColor( captureImage, captureImage_gray, CV_BGR2GRAY);//采集
的视频帧转化为灰度图

//【7】检测 SIFT 关键点并提取测试图像中的描述符
vector<KeyPoint> captureKeyPoints;
Mat captureDescription;
//【8】调用 detect 函数检测出特征关键点，保存在 vector 容器中
featureDetector.detect(captureImage_gray, captureKeyPoints);

//【9】计算描述符
featureExtractor.compute(captureImage_gray, captureKeyPoints,
captureDescription);

//【10】匹配和测试描述符，获取两个最邻近的描述符
Mat matchIndex(captureDescription.rows, 2, CV_32SC1),
matchDistance(captureDescription.rows, 2, CV_32FC1);
flannIndex.knnSearch(captureDescription, matchIndex,
matchDistance, 2, flann::SearchParams());//调用 K 邻近算法

//【11】根据劳氏算法 (Lowe's algorithm) 选出优秀的匹配
vector<DMatch> goodMatches;
for(int i = 0; i < matchDistance.rows; i++)
{
    if(matchDistance.at<float>(i, 0) < 0.6 *
matchDistance.at<float>(i, 1))
    {
        DMatch dmatches(i, matchIndex.at<int>(i, 0),
matchDistance.at<float>(i, 0));
        goodMatches.push_back(dmatches);
    }
}

//【12】绘制并显示匹配窗口
Mat resultImage;
drawMatches( captureImage, captureKeyPoints, srcImage,
keyPoints, goodMatches, resultImage);
imshow("匹配窗口", resultImage);

//【13】显示帧率
cout << "帧率= " << getTickFrequency() / (getTickCount() - time0)
<< endl;

//按下 ESC 键，则程序退出
if(char(waitKey(1)) == 27) break;
}

return 0;
}
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

此程序的运行截图如图 11.28、11.29 所示。

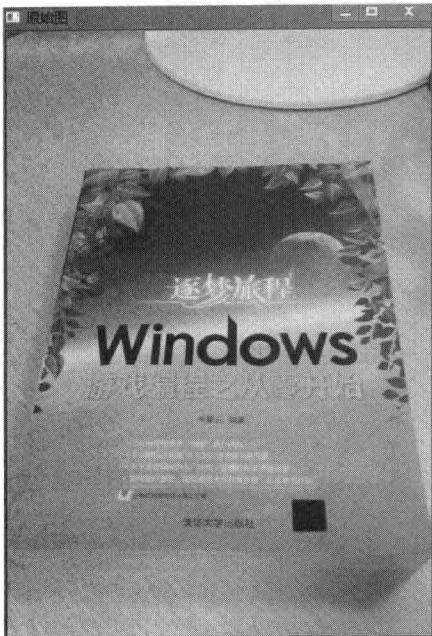


图 11.28 原始图

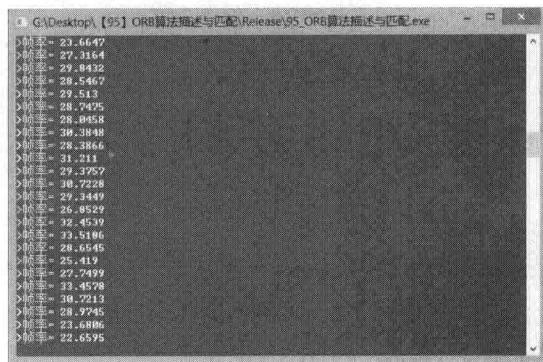


图 11.29 输出帧率的窗口

同样，当找一本和原图不相关的书籍对准摄像头，得到的匹配点寥寥无几。如图 11.30 所示。

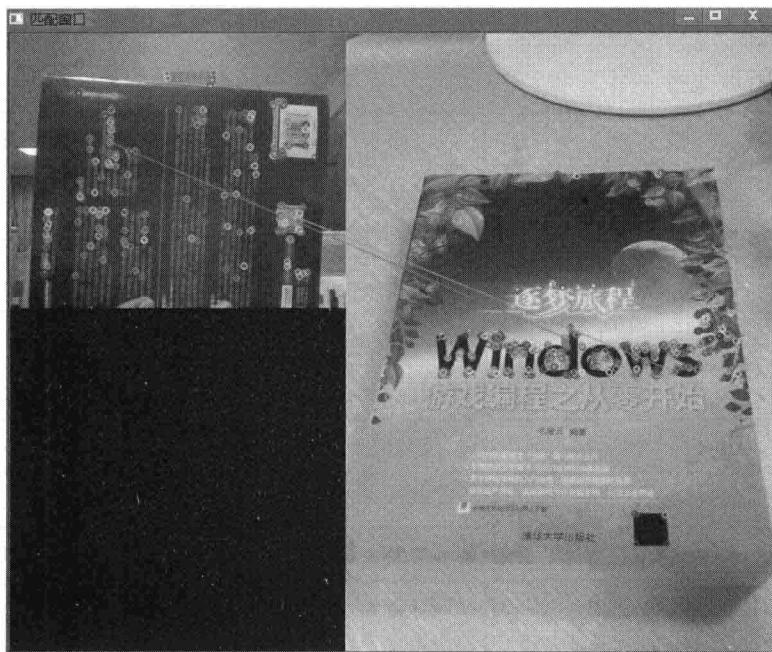


图 11.30 匹配点寥寥无几

而当找到原始图对应的书对准摄像头时，得到的匹配点多且整齐。如图 11.31 所示。

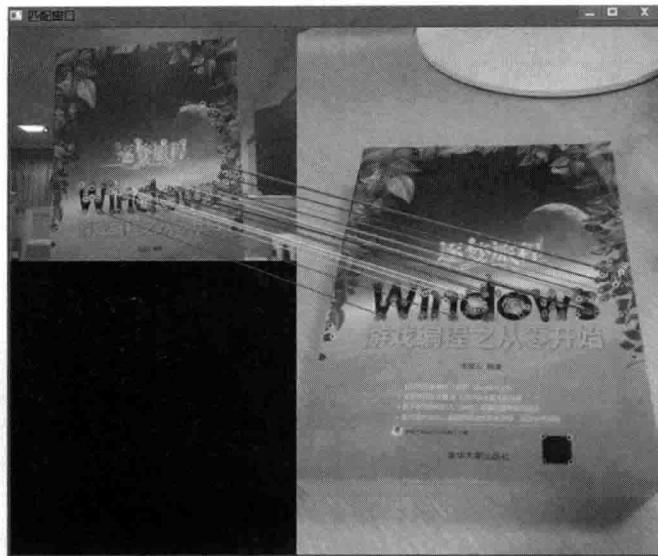


图 11.31 整齐的匹配点

## 11.6 本章小结

本章中我们主要使用 OpenCV2 讲解和实现了 SURF、SIFT 和 ORB 特征检测方法，其中 SURF 是尺度不变特征变换算法（SIFT 算法）的加速版，而 ORB 为 brief 算法的改进版。此外，还讲解了如何使用 FLANN 实现快速高效匹配。而在 FLANN 特征匹配的基础上，还可以进一步利用 Homography 映射找出已知物体。

### 本章核心函数清单

函数名称	说明	对应讲解章节
SURF 类、SurfFeatureDetector 类、SurfDescriptorExtractor 类	三者等价，同用于在 OpenCV 中进行 SURF 特征检测	11.1.3
drawKeypoints 函数	绘制关键点	11.1.4
drawMatches 函数	绘制出相匹配的两个图像的关键点	11.2.1
KeyPoint 类	用于表示特征点的信息	11.1.5
BruteForceMatcher 类	进行暴力匹配相关的操作	11.2.2
FlannBasedMatcher 类	实现 FLANN 特征匹配	11.3.1
DescriptorMatcher::match 函数	从每个描述符查询集中找到最佳匹配	11.3.2
findHomography 函数	找到并返回源图像和目标图像之间的透視变换 H	11.4.1
perspectiveTransform 函数	进行向量透視矩阵变换	11.4.2
ORB 类、OrbFeatureDetector 类、OrbDescriptorExtractor 类	三者等价，同用于在 OpenCV 中进行 ORB 特征检测	11.5.3

### 本章示例程序清单

示例程序序号	程序说明	对应章节
89	SURF 特征点检测	11.1.6
90	SURF 特征提取	11.2.3
91	使用 FLANN 进行特征点匹配	11.3.3
92	FLANN 结合 SURF 进行关键点的描述和匹配	11.3.4
93	SIFT 配合暴力匹配进行关键点描述和提取	11.3.5
94	寻找已知物体	11.4.3
95	利用 ORB 算法进行关键点的描述与匹配	11.5.4

作为一本入门级的 OpenCV 编程教材，我们以程序代码为主线，详细讲解了 OpenCV 最常用的 core、highgui、improc 和 feature2d 这四个组件的用法。至此，本书的主体内容已经结束。

愿大家在本书的帮助下，都能很好地入门和掌握新版 OpenCV。

愿本书能为新版 OpenCV 在国内的普及以及在世界范围内的发展，献上绵薄之力。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

# 附录

## A1 配套示例程序清单

本书含有 4 个部分 11 章，共有 95 个主线示例程序，为方便读者查阅和学习，总结成如下表格。

表 A1.1 配套示例程序清单

示例程序序号	程序说明	对应章节
1	OpenCV 环境配置的测试用例	1.3.8
2	快速上手 OpenCV 的第一个程序：图像显示	1.4.1
3	快速上手 OpenCV 的第二个程序：图像腐蚀	1.4.2
4	快速上手 OpenCV 的第三个程序：blur 图像模糊	1.4.3
5	快速上手 OpenCV 的第四个程序：canny 边缘检测	1.4.4
6	读取并播放视频	1.5.1
7	调用摄像头采集图像	1.5.2
8	官方例程引导、赏析之彩色目标跟踪：Camshift	2.1.1
9	官方例程引导、赏析之光流：optical flow	2.1.2
10	官方例程引导、赏析之点追踪：lkdemo	2.1.3
11	官方例程引导、赏析之人脸识别：objectDetection	2.1.4
12	官方例程引导、赏析之支持向量机：支持向量机引导	2.1.5
13	官方例程引导、赏析之支持向量机：处理线性不可分数据	2.1.5
14	printf 函数的用法示例	2.6.2
15	用 imwrite 函数生成 png 透明图	3.1.8
16	综合示例程序：图像的载入、显示与输出	3.1.9
17	为程序界面添加滑动条	3.2.1
18	鼠标操作示例	3.3
19	基础图像容器 Mat 类的使用	4.1.7
20	用 OpenCV 进行基本绘图	4.3
21	操作图像中像素的方法一：用指针访问像素	5.1.5、5.1.6
22	操作图像中像素的方法二：用迭代器操作像素	5.1.5、5.1.6

续表

示例程序序号	程序说明	对应章节
23	操作图像中像素的方法三：动态地址计算	5.1.5、5.1.6
24	遍历图像中像素的 14 种方法	5.1.6
25	初级图像混合	5.2.4
26	多通道图像混合	5.3.3
27	图像对比度、亮度值调整	5.4.3
28	离散傅里叶变换	5.5.8
29	XML 和 YAML 文件的写入	5.6.3
30	XML 和 YAML 文件的读取	5.6.4
31	方框滤波：boxFilter 函数的使用	6.1.11
32	均值滤波：blur 函数的使用	6.1.11
33	高斯滤波：GaussianBlur 函数的使用	6.1.11
34	综合示例：图像线性滤波	6.1.12
35	中值滤波：medianBlur 函数的使用	6.2.4
36	双边滤波：bilateralFilter 函数的使用	6.2.4
37	综合示例：图像滤波	6.2.5
38	膨胀：dilate 函数的使用	6.3.5
39	腐蚀：erode 函数的使用	6.3.5
40	综合示例：腐蚀与膨胀	6.3.6
41	用 morphologyEx() 函数实现形态学膨胀	6.4.8
42	用 morphologyEx() 函数实现形态学腐蚀	6.4.8
43	用 morphologyEx() 函数实现形态学开运算	6.4.8
44	用 morphologyEx() 函数实现形态学闭运算	6.4.8
45	用 morphologyEx() 函数实现形态学梯度	6.4.8
46	用 morphologyEx() 函数实现形态学“顶帽”	6.4.8
47	用 morphologyEx() 函数实现形态学“黑帽”	6.4.8
48	综合示例：形态学滤波	6.4.9
49	漫水填充算法：floodFill 函数	6.5.3
50	综合示例：漫水填充	6.5.4
51	尺寸调整：resize() 函数的使用	6.6.5

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

续表

示例程序序号	程序说明	对应章节
52	向上采样图像金字塔: pyrUp()函数的使用	6.6.6
53	向下采样图像金字塔: pyrDown()函数的使用	6.6.6
54	综合示例: 图像金字塔与图片尺寸缩放	6.6.7
55	示例程序: 基本阈值操作	6.7.3
56	Canny 边缘检测	7.1.2
57	Sobel 算子的使用	7.1.3
58	Laplacian 算子的使用	7.1.4
59	Scharr 滤波器	7.1.5
60	综合示例: 边缘检测	7.1.6
61	标准霍夫变换: HoughLines()函数的使用	7.2.4
62	累计概率霍夫变换: HoughLinesP()函数	7.2.5
63	霍夫圆变换: HoughCircles()函数	7.2.8
64	综合示例: 霍夫变换	7.2.9
65	实现重映射: remap()函数	7.3.3
66	综合示例程序: 实现多种重映射	7.3.4
67	仿射变换	7.4.5
68	直方图均衡化	7.5.3
69	轮廓查找	8.1.3
70	查找并绘制轮廓	8.1.4
71	凸包检测基础	8.2.3
72	寻找和绘制物体的凸包	8.2.4
73	创建包围轮廓的矩形边界	8.3.6
74	创建包围轮廓的圆形边界	8.3.7
75	使用多边形包围轮廓	8.3.8
76	图像轮廓矩	8.4.4
77	分水岭算法的使用	8.5.2
78	实现图像修补	8.6.2
79	H-S 二维直方图的绘制	9.2.3
80	一维直方图的绘制	9.2.4

续表

示例程序序号	程序说明	对应章节
81	RGB 三色直方图的绘制	9.2.5
82	直方图对比	9.3.2
83	反向投影	9.4.7
84	模板匹配	9.5.3
85	实现 Harris 角点检测：cornerHarris()函数的使用	10.1.4
86	harris 角点检测与绘制	10.1.5
87	Shi-Tomasi 角点检测	10.2.3
88	亚像素级角点检测	10.3.3
89	SURF 特征点检测	11.1.6
90	SURF 特征提取	11.2.3
91	使用 FLANN 进行特征点匹配	11.3.3
92	FLANN 结合 SURF 进行关键点的描述和匹配	11.3.4
93	SIFT 配合暴力匹配进行关键点描述和提取	11.3.5
94	寻找已知物体	11.4.3
95	利用 ORB 算法进行关键点的描述与匹配	11.5.4

## A2 随书额外附赠的程序一览

附录的这一部分算是购买本书后获赠的福利。很清楚大家会对目前稀缺的基于 OpenCV2 的这余下的二十几个示例程序抱有很大的兴趣，于是便有了这一部分的诞生。

本书在开始策划写作时，准备了一百多个基于 OpenCV2 的示例程序作为教学素材，最终定下来正文部分采用的示例程序是 95 个，也就是现在大家在书本中已经看到的一句一句串起本书主线内容那些“中流砥柱”们。

这样余下来的这些示例程序，也就找到了他们存在的意义。

需要说明的是，这些程序大多取材于 OpenCV 官方示例程序、或者取材、灵感源自于参考文献中列举出的国外的 OpenCV2 教程，也很容易转化到 OpenCV3 中运行。这些程序位于书本 OpenCV2 版的示例程序包中“【2】附赠示例程序源代码”文件夹里，并按序号顺序进行了标注。

为了方便大家查阅和学习，如下便是对这些附赠程序的表格式总结清单：

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

表 A2.1 附赠程序清单

程序序号	示例名称	说明
1	随机图形和文字生成示例 (randomtext)	此程序利用 OpenCV 中的各种绘制函数随机生成图形和文字，有一定的学习和研究价值
2	生成彩色色条 (gencolors)	用法 generateColors 函数生成彩色色条并进行显示
3	卡尔曼滤波 (kalman)	用 OpenCV 动态绘制卡尔曼滤波，运行程序后可直接得出动画效果。用键盘任意按键重置轨迹并更新速度。使用 ESC 键结束程序
4	渐变过渡各种图形滤波 (median_blur)	渐变过渡效果的各种图形滤波的显示，并输出说明性文字到窗口中
5	距离变换 (distanceTransform)	此程序用于演示边缘图像之间的距离变换。 按键说明： <b>【ESC】</b> -退出程序 <b>【c】</b> -使用 C/Inf 度量 <b>【l】</b> -使用 L1 度量 <b>【2】</b> -使用 L2 度量 <b>【3】</b> - 使用 $3 \times 3$ 的掩膜 <b>【5】</b> - 使用 $5 \times 5$ 的掩膜 <b>【0】</b> - 采用精确的距离变换 <b>【v】</b> - 切换到 Voronoi 图 (Voronoi diagram) 模式 <b>【p】</b> - 切换到基于像素的 Voronoi 图模式 <b>【SPACE】</b> - 在各种模式间切换
6	把图像映射到极指数空间 (Log Polar)	此程序用于把图像映射到极指数空间，操作说明如下： <b>【n】</b> -采用最邻近像素技术 (nearest pixel technique) <b>【b】</b> -采用双线性插值技术 (bilinear interpolation technique) <b>【o】</b> -使用重叠的圆形的接受域 (overlapping circular receptive fields) <b>【a】</b> -使用相邻的接受域 (adjacent receptive fields)
7	filter2D 滤波器的用法	用 OpenCV 中的 filter2D 滤波器来模糊一张图片，并将结果存储到 “filtered_image.jpg” 中
8	grabCut 图像分割示例	此程序演示了 OpenCV 中 GrabCut 图像分割的使用。程序运行后，我们需要用鼠标圈出需要分割的那部分物体。按键说明如下： <b>【ESC】</b> -退出程序 <b>【r】</b> -恢复原始图片 <b>【n】</b> -开始迭代，和进行下一次迭代 <b>【鼠标左键】</b> -设置选中矩形区域 <b>【Ctrl+鼠标左键】</b> -设置 GC_BGD 像素 <b>【Shift+鼠标左键】</b> -设置 CG_FGD 像素 <b>【Ctrl+鼠标右键】</b> -设置 GC_PR_BGD 像素 <b>【Shift+鼠标右键】</b> -设置 CG_PR_FGD 像素

续表

程序序号	示例名称	说明
9	MeanShift 图像分割示例	此程序演示了 OpenCV 中 MeanShift 图像分割的使用。程序运行后我们可以通过 3 个滑动条调节分割效果。3 个滑动条代表的参数分别为空间窗的半径 (spatialRad)、色彩窗的半径 (colorRad)、最大图像金字塔级别 (maxPyrLevel)
10	用滑动控制图像直方图	此程序结合滚动条的创建，演示了如何用 calcHist 来创建直方图。可以条件滚动条，看到不同形态的图像直方图
11	找到图像最小的封闭轮廓	此程序结合了轮廓查找和多边形曲线精度逼近，来演示如何找到图像最小的封闭轮廓。运行程序即可观察出最终效果
12	Retina 特征点检测	此程序用于演示 Retina 特征点检测，运行后会得到多幅运行效果图
13	摄像头帧数检测	此程序非常简单实用，用于调用摄像头采集图像，并显示当前采集的图像帧数
14	视频截图	此程序也是非常简单实用，用于读取视频并播放，在播放时，按下【Space】空格键可以截图，图片将存放在工程目录下，而【Esc】和【q】键可以退出程序
15	对视频的快速角点检测	此程序用于演示如何对视频进行快速角点检测。按键说明如下： 【t】-抓取一个引用帧的进行匹配 【l】-使引用更新每一帧视频 【q】或【ESC】-退出程序
16	视频简单色彩检测	此程序调用摄像头进行视频采集，输出实时帧率，进行简单色彩检测，并可以用滑动条控制 R、G、B 三个通道的高低阈值
17	跟踪分割视频中运动的物体	此程序演示了一种寻找轮廓，连接组件，清除背景的简单方法，实现跟踪分割视频中运动的物体。程序运行开始后，便开始“学习背景”，我们可以通过【Space】空格键来切换是否打开“背景学习”技术
18	视频的直方图反向投影	此程序用摄像头采集视频，并进行实时的直方图方向投影显示
19	计算视频中两个图像区域的相似度	此程序用摄像头采集视频，然后我们可以在视频上用鼠标选定两个矩形区域，然后 OpenCV 就会为我们算出图像区域的相似度数值，并绘制出 RGB 三色直方图
20	视频前后背景分离	此程序展示了视频前后背景分离的方法，程序首先会“学习背景”，然后进行分割。可以用过【Space】空格进行功能切换

续表

程序序号	示例名称	说明
21	用高斯背景建模分离背景	此程序展示了用高斯背景建模进行视频的背景分离方法，程序首先会“学习背景”，然后进行分割。可以用过【Space】空格进行功能切换

### A3 书本核心函数清单

附录的这一部分，为每章（除1、2两章）的末尾小节中“本章核心函数清单”的一个汇总，也可以说是本书讲解的所有OpenCV核心函数和类的汇总。同样的，在此处进行列举，以方便读者在需要的时候进行查阅。

表 A3.1 本书核心函数/类清单

函数/类名称	用途	讲解章节
imread	用于读取文件中的图片到OpenCV中	3.1.4
imshow	在指定的窗口中显示一幅图像	3.1.5
namedWindow	用于创建一个窗口	3.1.7
imwrite	输出图像到文件	3.1.8
createTrackbar	用于创建一个可以调整数值的轨迹条	3.2.1
getTrackbarPos	用于获取轨迹条的当前位置	3.2.2
SetMouseCallback	为指定的窗口设置鼠标回调函数	3.3
Mat::Mat()	Mat类的构造函数	4.1.4
Mat::Create()	Mat类的成员函数，可用于Mat类的初始化操作	4.1.4
Point类	用于表示点的数据结构	4.2.1
Scalar类	用于表示颜色的数据结构	4.2.2
Size类	用于表示尺寸的数据结构	4.2.3
Rect类	用于表示矩形的数据结构	4.2.4
CvtColor()	用于颜色空间转换	4.2.5
line	绘制直线	4.3.4
ellipse	绘制椭圆	4.3.1
rectangle	绘制矩形	4.3.5
circle	绘制圆	4.3.2
fillPoly	绘制填充的多边形	4.3.3
addWeighted	计算两个数组（图像阵列）的加权和	5.2.3

续表

函数/类名称	用途	讲解章节
split	将一个多通道数组分离成几个单通道数组	5.3.1
merge	将多个数组组合合并成一个多通道的数组	5.3.2
dft	对一维或二维浮点数数组进行正向或反向离散傅里叶变换	5.5.2
getOptimalDFTSize	返回给定向量尺寸的傅里叶最优尺寸大小	5.5.3
copyMakeBorder	扩充图像边界	5.5.4
magnitude	计算二维矢量的幅值	5.5.5
log	计算每个数组元素绝对值的自然对数	5.5.6
normalize	进行矩阵归一化	5.5.7
FileStorage 类	进行文件操作的类	5.6.2
boxFilter	使用方框滤波来模糊一张图片	6.1.11
blur	对输入的图像进行均值滤波操作	6.1.11
GaussianBlur	用高斯滤波器来模糊一张图片	6.1.11
medianBlur	使用中值滤波器来模糊一张图片	6.2.4
bilateralFilter	用双边滤波器来模糊处理一张图片	6.2.4
dilate	使用像素邻域内的局部极大运算符来膨胀一张图片	6.3.5
erode	使用像素邻域内的局部极小运算符来腐蚀一张图片	6.3.5
morphologyEx	利用基本的膨胀和腐蚀技术, 来执行更加高级形态学变换, 如开闭运算, 形态学梯度、顶帽、黑帽等, 也可以实现最基本的图像膨胀和腐蚀	6.4.7
floodFill	用指定的颜色从种子点开始填充一个连接域, 实现漫水填充算法	6.5.3
pyrUp	向上采样并模糊一张图片, 说白了就是放大一张图片	6.6.6
pyrDown	向下采样并模糊一张图片, 说白了就是缩小一张图片	6.6.6
Threshold	对单通道数组应用固定阈值操作	6.7.1
adaptiveThreshold	对矩阵采用自适应阈值操作	6.7.2
Canny	利用 Canny 算子来进行图像的边缘检测	7.1.2
Sobel	使用扩展的 Sobel 算子, 来计算一阶、二阶、三阶或混合图像差分	7.1.3
Laplacian	计算出图像经过拉普拉斯变换后的结果	7.1.4

续表

函数/类名称	用途	讲解章节
Scharr	使用 Scharr 滤波器运算符计算 x 或 y 方向的图像差分	7.1.5
HoughLines	找出采用标准霍夫变换的二值图像线条	7.2.4
HoughLinesP	采用累计概率霍夫变换（PPHT）来找出二值图像中的直线	7.2.5
HoughCircles	利用霍夫变换算法检测出灰度图中的圆	7.2.8
remap	根据指定的映射形式，将源图像进行重映射几何变换	7.3.2
warpAffine	依据公式对图像做仿射变换	7.4.3
getRotationMatrix2D	计算二维旋转变换矩阵	7.4.4
equalizeHist	实现图像的直方图均衡化	7.5.2
findContours	在二值图像中寻找轮廓	8.1.1
drawContours	在图像中绘制外部或内部轮廓	8.1.2
convexHull	寻找图像点集中的凸包	8.2.2
boundingRect	计算并返回指定点集最外面（up-right）的矩形边界	8.3.1
minAreaRect	寻找可旋转的最小面积的包围矩形	8.3.2
minEnclosingCircle	利用一种迭代算法，对给定的 2D 点集，寻找面积最小的可包围他们的圆形	8.3.3
fitEllipse	用椭圆拟合二维点集	8.3.4
approxPolyDP	用指定精度逼近多边形曲线	8.3.5
moments	计算多边形和光栅形状的最高达三阶的所有矩	8.4.1
contourArea	计算整个轮廓或部分轮廓的面积	8.4.2
arcLength	计算封闭轮廓的周长或曲线的长度	8.4.3
watershed	实现分水岭算法	8.5.1
inpaint	进行图像修补，从扫描的照片中清除灰尘和划痕，或者从静态图像或视频中去除不需要的物体	8.6.1
calcHist	计算一个或者多个阵列的直方图	9.2.1
minMaxLoc	在数组中找到全局最小值和最大值	9.2.2
compareHist	对两幅直方图进行比较	9.3.1
calcBackProject	计算直方图的反向投影	9.4.5
mixChannels	由输入参数拷贝某通道到输出参数特定的通道中	9.4.6
matchTemplate	匹配出和模板重叠的图像区域	9.5.2

续表

函数/类名称	用途	讲解章节
CornerHarris	运行 Harris 角点检测算子来进行角点检测	10.1.4
goodFeaturesToTrack	结合 Shi-Tomasi 算子确定图像的强角点	10.2.2
cornerSubPix	寻找亚像素角点位置	10.3.2
SURF 类、SurfFeatureDetector 类、SurfDescriptorExtractor 类	三者等价，同用于在 OpenCV 中进行 SURF 特征检测	11.1.3
drawKeypoints	绘制关键点	11.1.4
drawMatches	绘制出相匹配的两个图像的关键点	11.2.1
KeyPoint 类	用于表示特征点的信息	11.1.5
BruteForceMatcher 类	进行暴力匹配相关的操作	11.2.2
FlannBasedMatcher 类	实现 FLANN 特征匹配	11.3.1
DescriptorMatcher::match	从每个描述符查询集中找到最佳匹配	11.3.2
findHomography	找到并返回源图像和目标图像之间的透视变换 H	11.4.1
perspectiveTransform	进行向量透视矩阵变换	11.4.2
ORB 类、OrbFeatureDetector 类、OrbDescriptorExtractor 类	三者等价，同用于在 OpenCV 中进行 ORB 特征检测	11.5.3

## A4 Mat 类函数一览

Mat 类作为新版 OpenCV 中最核心的数据结构，有着非常健全的代码构造。本书正文第四章中已经对 Mat 类常用的一些构造函数和成员函数有了一些介绍，而这里的这一部分是一个补充，会将剩下遗漏的 Mat 类的构造函数、析构函数和成员函数进行完整的介绍。

### A4.1 构造函数：Mat::Mat

Mat 类的众多构造函数。

```
C++: Mat::Mat()
C++: Mat::Mat(int rows, int cols, int type)
C++: Mat::Mat(Size size, int type)
C++: Mat::Mat(int rows, int cols, int type, const Scalar& s)
C++: Mat::Mat(Size size, int type, constScalar& s)
C++: Mat::Mat(const Mat& m)
C++: Mat::Mat(int rows, int cols, int type, void* data, size_t
step=AUTO_STEP)
C++: Mat::Mat(Size size, int type, void*data, size_t step=AUTO_STEP)
C++: Mat::Mat(const Mat& m, constRange& rowRange, const Range& colRange)
```

```

C++: Mat::Mat(const Mat& m, constRect& roi)
C++: Mat::Mat(const CvMat* m, boolcopyData=false)
C++: Mat::Mat(const IplImage* img, boolcopyData=false)
C++: template<typename T, int n>explicit Mat::Mat(const Vec<T, n>& vec,
bool copyData=true)
C++: template<typename T, int m, intn> explicit Mat::Mat(const Matx<T,
m, n>& vec, bool copyData=true)
C++: template<typename T> explicitMat::Mat(const vector<T>& vec, bool
copyData=false)
C++: Mat::Mat(const MatExpr& expr)
C++: Mat::Mat(int ndims, const int* sizes,int type)
C++: Mat::Mat(int ndims, const int* sizes,int type, const Scalar& s)
C++: Mat::Mat(int ndims, const int* sizes,int type, void* data, const
size_t* steps=0)
C++: Mat::Mat(const Mat& m, constRange* ranges)

```

这些构造函数的参数含义列举如下：

表 A3.1 Mat 类构造函数参数列举

参数	含义
ndims	数组的维数
rows	2 维数组的行数
cols	2 维数组的列数
size	2 维数组的尺寸 Size(cols, rows) . 在 Size() 构造函数中行数和列数在顺序上为反转过来的
sizes	指定 n 维数组形状的整数数组
type	数组的类型。使用 CV_8UC1, ... ..., 创建 1-4 通道的矩阵, CV_64FC4 或 CV_8UC(n), ... ..., CV_64FC(n) 可以创建多通道 (高达 CV_MAX_CN 通道) 矩阵
s	一个可选的初始化每个矩阵元素的参数。要在矩阵建成后将所有元素设置为特定值可以用 Mat 的赋值运算符 Mat::operator=(constScalar& value)
data	指向用户数据的指针。矩阵构造函数传入 data 和 step 参数不分配矩阵数据。相反, 它们只是初始化矩阵头指向指定的数据, 这意味着没有数据的复制。此操作是很高效的, 可以用来处理使用 OpenCV 函数的外部数据。外部数据不会自动释放, 所以你应该小心处理它 step – 每个矩阵行占用的字节数。如果任何值应包括每行末尾的填充字节。如果缺少此参数 (设置为 AUTO_STEP), 假定没有填充和实际的步长用 cols*elemSize() 计算。 请参阅 Mat::elemSize()
steps	多维数组 (最后一步始终设置为元素大小) 的情况下的 ndims-1 个步长的数组。如果没有指定的话, 该矩阵假定为连续

续表

参数	含义
M	分配给构造出来的矩阵的阵列（作为一个整体或部分）。这些构造函数没有复制数据。相反，指向 m 的数据或它的子数组的头被构造并被关联到 m 上。引用计数器中无论如何都将递增。所以，当您修改矩阵的时候，自然而然就使用了这种构造函数，您还修改 m 中的对应元素。如果你想要独立的子数组的副本，请使用 Mat::clone()
img	指向老版本的 IplImage 图像结构的指针。默认情况下，原始图像和新矩阵之间共享数据。但当 copyData 被设置时，完整的图像数据副本就创建起来了
vec	矩阵的元素构成的 STL 向量。矩阵可以取出单独一列并且该列上的行数和矢量元素的数目相同。矩阵的类型匹配的向量元素的类型。构造函数可以处理任意的有正确声明的 DataType 类型。这意味着矢量元素不支持的混合型结构，它们必须是数据(numbers)原始数字或单型数值元组。对应的构造函数是显式的。由于 STL 向量不会自动转换为 Mat 实例，您应显式编写 Mat(vec)。除非您将数据复制到矩阵 (copyData = true)，没有新的元素被添加到向量中，因为这样可能会造成矢量数据重新分配，并且因此使得矩阵的数据指针无效
copyData	指定 STL 向量或旧型 CvMat 或 IplImage 是应复制到(true)新构造的矩阵中还是(false)与之共享基础数据的标志，复制数据时，使用 Mat 引用计数机制管理所分配的缓冲区。虽然数据共享的引用计数为 NULL，但是分配数据必须在矩阵被析构之后才可以释放
rowRange	矩阵阵列 m 之行数的取值范围。另外，可以使用 Range::all() 来取所有的行
colRange	M 列数的取值范围。使用 Range::all() 来取所有的列
ranges	表示 M 沿每个维度选定的区域的数组
expr	矩阵表达式。具体请参见上文“矩阵表达式”一节的内容

以上这些都是 Mat 生成一个矩阵的各类构造函数。其实很多时候，默认构造函数就足够了，其可由 OpenCV 函数来分配数据空间。而通过 Mat::create()，构造的矩阵可以进一步分配给其他的矩阵或者矩阵表达式。

## A4.2 析构函数 Mat::~Mat

Mat 的析构函数为 Mat::~Mat()。原型声明如下：

```
C++: Mat::~Mat()
```

Mat 类的析构函数其实就是调用成员函数 Mat::release() 进行析构操作。

## A4.3 Mat 类成员函数

由于 Mat 类的类成员函数众多，不妨让我们通过列表的方式来进行列举和说明：

表 A3.2 Mat 类的类成员函数一览

Mat 类成员函数	作用
Mat::operator =	提供矩阵赋值操作
Mat::row	创建一个指定行数的矩阵头
Mat::col	创建一个指定列数的矩阵头
Mat::rowRange	创建指定行跨度 (span) 的矩阵头
Mat::colRange	创建指定列跨度的矩阵头
Mat::diag	提取或创建矩阵对角线
Mat::clone	创建一个数组及其基础数据的完整副本
Mat::copyTo	把矩阵复制到另一个矩阵中
Mat::convertTo	在缩放或不缩放的情况下转换为另一种数据类型
Mat::assignTo	提供了一种 convertTo 的功能形式
Mat::setTo	将阵列中所有的或部分的元素设置为指定的值
Mat::reshape	在无需复制数据的前提下改变 2D 矩阵的形状和通道数或其中之一
Mat::t	通过矩阵表达式 (matrix expression) 实现矩阵的转置
Mat::inv	反转矩阵
Mat::mul	执行两个矩阵按元素相乘或这两个矩阵的除法
Mat::cross	计算 3 元素向量的一个叉乘积
Mat::dot	计算两向量的点乘
Mat::zeros	返回指定的大小和类型的零数组
Mat::ones	返回一个指定的大小和类型的全为 1 的数组
Mat::eye	返回一个恒等指定大小和类型矩阵
Mat::create	分配新的阵列数据
Mat::addrf	计数器参考
Mat::release	在必要的情况下，递减引用计数并释放该矩阵
Mat::resize	更改矩阵的行数
Mat::reserve	保留一定数量的行空间
Mat::push_back	将元素添加到矩阵的底部
Mat::pop_back	从底部的列表中删除元素
Mat::locateROI	父矩阵内定位矩阵头
Mat::adjustROI	调整子阵大小及其在父矩阵中的位置

续表

Mat 类成员函数	作用
Mat::operator()	提取矩形子阵
Mat::operator CvMat	创建矩阵 CvMat 头
Mat::operator IplImage	创建 IplImage 矩阵头
Mat::isContinuous	返回矩阵是否连续
Mat::elemSize	返回矩阵元素大小（以字节为单位）
Mat::type	返回一个矩阵元素的类型
Mat::depth	返回一个矩阵元素的深度
Mat::channels	返回矩阵通道的数目
Mat::step1	返回矩阵归一化的第一步
Mat::size	返回一个矩阵大小
Mat::empty	如果数组中没有元素，则返回 true
Mat::ptr	返回指定矩阵行的指针
Mat::at	返回对指定数组元素的引用
Mat::begin	返回矩阵迭代器，并将其设置为第一个矩阵元素
Mat::end	返回矩阵迭代器，并将其设置在最后一个元素之后 (after-last)

可以发现，Mat 类的成员函数众多，可以实现各种各样的操作。大家需要对 Mat 类进行有关操作的时候，可以在此表中进行查阅。

至此，本书的所有内容都已经结束。

# 主要参考文献

- [1] Rafael C.Gonzalez, Digital Image Processing Third Edition [M], Prentice Hall, 2007
- [2] Richard Szeliski,Computer Vision Algorithms and Applications[M], Springer, 2010
- [3] Adrian Kaehler, Learning OpenCV, oreilly, 2008
- [4] Daniel Lélis Baggio,Mastering OpenCV with Practical Computer Vision Projects, Packt Publishing, 2012
- [5] Robert Laganière ,OpenCV 2 Computer Vision Application Programming Cookbook, Packt Publishing,2011
- [6] Samarth Brahmbhatt ,Practical OpenCV,APress,2013
- [7] OpenCV 2.4.9.0 documentation, <http://docs.opencv.org/modules/refman.html>, 2014

## 读者热评

很好，文章透彻，程序易懂，一看就会，能够快速掌握。

——枫羽落雪

对于新版OpenCV能有这么透彻的讲解，真是太好了。

——yuhangrenmin

文笔不错，而且对编程的理解很到位，向你学习。再次感谢。

——vacuum8899

很感谢你，看了你写的教程让我少走了很多弯路！

——nicepainkiller

您写的真透彻！解决很多的疑问和麻烦！感动！

——cy\_543

写的很好，深入浅出，给的程序也很实用，佩服。

——xuliangfirst

代码写的通俗易懂，对于新手来说真是很有帮助！真心感谢楼主的无私奉献！

——Alex

作者代码写得很规范，注释也很清晰，结合书本讲解，浅显易懂。

——SkyHeight

讲得挺全面的。原理、实现、具体代码，非常适合初学者实用的教程。

——gmf\_henu

文章讲解、注释内容写的很详细，初学者都能很容易看懂。

——mcuc51

代码详细具体，注释明确，很好的学习资料，参考书本学习，效果加倍。

——tuling56

相当感谢，写的不错，对于入门的初学者非常不错。

——lqiang0630

怒赞，写代码从细节写起，看这样的代码顺眼！

——liuchyi



博文视点Broadview



@博文视点Broadview

上架建议：计算机>图像处理

ISBN 978-7-121-25331-7



9 787121 253317

定价：79.00元



责任编辑：陈晓猛  
封面设计：李玲