Proposed modification for EBImage 3.10

Keraudren Kevin
kevin.keraudren10@imperial.ac.uk

January 2012

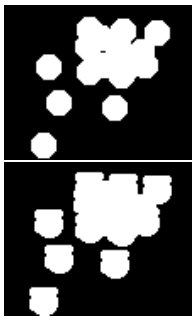
Contents

1 Bugs

1.1 Disallowing even sizes for kernels in `makeBrush`

The `makeBrush` currently allows odd and even size kernels. Dilation with *disc* of even size creates artifacts. Thus, even sizes should not be allowed.

```
> x = readImage(system.file("images", "shapes.png", package="EBImage"))
> x = x[376:486,146:236]
> kern = makeBrush(11, shape='disc')
> display(dilate(x, kern), title='Dilation of x')
> kern = makeBrush(12, shape='disc')
> display(dilate(x, kern), title='Dilation of x')
```



Dilation with an odd and even size kernel.

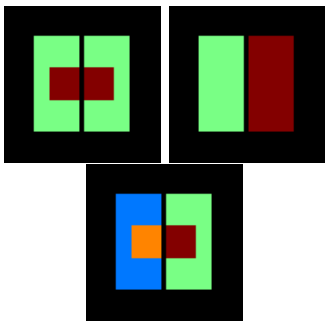
Proposed fix in `EBImage/R/morphology.R`:

```
if ( (size %% 2) != 1) stop("'size' must be an odd number")
```

1.2 Connected objects wrongly merged by `bwlabel`

The function `bwlabel` currently merges parts which are not separated by 0 regions. This is infortunate when we split an object and whish to label separately each parts, without merging them with adjacent objects. The function `colorize` used below is presented in Section 3.6.

```
> img = Image(dim=c(100,100))
> img[20:80,20:80] = 1
> img[30:70,40:60] = 2
> img[49:51,] = 0
> display(colorize(img))
> display(colorize(bwlabel(img)))
```



Top-left image is the input, top-right image is the wrong output of `bwlabel`,
bottom image is the correct output of `bwlabel`.

Proposed fix in EBImage/src/floodFill.cpp:

```
// assuming binary images: 0 is background and everything else foreground
// foreground is negated
for (i=0; i<nz*size.x*size.y; i++) {
  if (REAL(res)[i]!=0.0) REAL(res)[i]*=-1;
}
...
if (REAL(res)[kx + ky*size.x + i*size.x*size.y] < 0) {
```

Note: the old behaviour can be reproduced using

```
> bwlabel(img>0)
```

2 Code optimization

2.1 <*deque*> instead of <*list*> in `watershed`

From http://www.cplusplus.com/reference/stl/deque/:

"For operations that involve frequent insertion or removals of elements at positions other than the beginning or the end, deques perform worse and have less consistent iterators and references than lists."

The current code for `watershed` uses <*list*> although elements are inserted or removed only at the ends. Using <*deque*> would be more efficient.

```
> x = readImage(system.file('images', 'shapes.png', package='EBImage'))
> x = resize(x, 1000,1000)
> y = distmap(x)
> system.time(watershed(y))
```

User time:

<*list*>	<*deque*>
60.06 *s*	2.610 *s*

Proposed fix in EBImage/src/watershed.cpp:

```
/* deque of STL, C++ */
#include <deque>
...
typedef std::deque<int>     IntList;
typedef std::deque<TheSeed> SeedList;
```

2.2 *for* loops instead of FFT for 2D convolution

For 2D convolution, simple *for* loops, which is what is being done in the biOps package, are faster than the FFT.

```
> sobel_filter2 = function( img ) {
   data = c(1,0,-1,2,0,-2,1,0,-1)
   kernelX = matrix(data, 3, 3, byrow = TRUE)
   kernelY = matrix(data, 3, 3, byrow = FALSE)
   x = filter2(img, kernelX)
   y = filter2(img, kernelY)
   return( sqrt(x^2 + y^2) )
 }
> sobel_imageConvolve = function( img ) {
   data = c(1,0,-1,2,0,-2,1,0,-1)
   kernelX = matrix(data, 3, 3, byrow = TRUE)
   kernelY = matrix(data, 3, 3, byrow = FALSE)
   x = imageConvolve(img, kernelX)
   y = imageConvolve(img, kernelY)
   return( sqrt(x^2 + y^2) )
 }
> x = readImage(system.file('images', 'lena.gif', package='EBImage'))
> x = resize(x, 1000,1000)
> system.time(sobel_filter2(x))


   user  system elapsed
  2.870   0.210   3.097


> system.time(sobel_imageConvolve(x))


   user  system elapsed
  0.380   0.020   0.395
```

Output of `sobel_filter2` and `sobel_imageConvolve`.

Proposed fix: see in EBImage/src/drawable.c:

3 Additions

3.1 separate

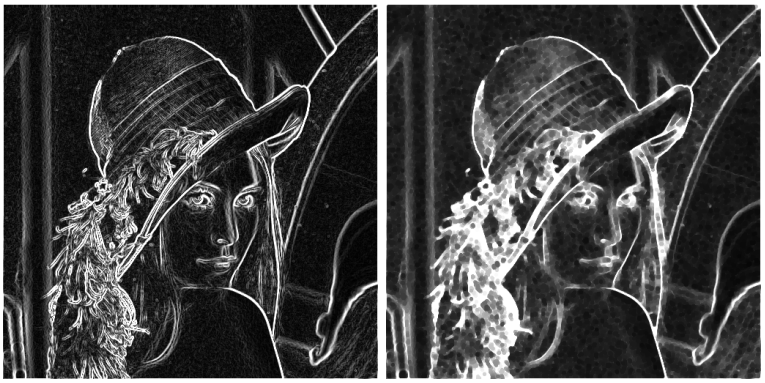
Function to disconnect touching objects.

```
> x = readImage(system.file('images', 'shapes.png', package='EBImage'))
> x = x[110:512,1:130]
> if (interactive()) display(x, title='Binary')
> y = distmap(x)
> if (interactive()) display(normalize(y), title='Distance map')
> w = watershed(y)
> if (interactive()) display(normalize(w), title='Watershed')
> z = separate(w)
> if (interactive()) display(normalize(z), title='Separate')
```

See code in EBImage/src/objects.c

3.2 Grayscale morphological operations

```
> x = readImage(system.file("images", "lena.gif", package="EBImage"))
> if (interactive()) display(x, title='Lena')
> # Sobel filter for edge detection
> sobel = function( img ) {
 data = c(1,0,-1,2,0,-2,1,0,-1)
 kernelX = matrix(data, 3, 3, byrow = TRUE)
 kernelY = matrix(data, 3, 3, byrow = FALSE)
 x = imageConvolve(img, kernelX)
 y = imageConvolve(img, kernelY)
 return( sqrt(x^2 + y^2) )
 }
> y = sobel(x)
> y2 = closing(y,kern=makeBrush(5, shape='disc'),binary=FALSE)
> if (interactive()) display(y, title='Sobel lena')
> if (interactive()) display(y2, title='Closed Sobel lena')
```



See code in EBImage/src/morphology.c

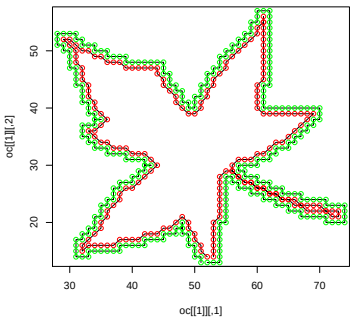
3.3 drawPolyline

```
> img = readImage(system.file("images", "shapes.png", package="EBImage"))
> img = img[110:512,1:130]
> labels = bwlabel(img)
> oc = ocontour(labels)
> poly = list()
> for ( i in 1:max(labels) ) {
 x = filter(oc[[i]][,1],rep(1,21)/21,method='convolution',circular=TRUE)
 y = filter(oc[[i]][,2],rep(1,21)/21,method='convolution',circular=TRUE)
 p = cbind(x,y)
 p = rbind(p,p[1,])
 poly[[i]] = p
 }
> output = rgbImage(red=img,green=img,blue=img)
> output = drawPolyline(output,poly,stroke.color='red')
> if (interactive()) display(output)
```

See code in EBImage/src/drawable.c

3.4 External contours in ocontour

```
> x = readImage(system.file("images", "shapes.png", package="EBImage"))
> x = x[1:120,50:120]
> oc = ocontour(x)
> plot(oc[[1]], type='l')
> points(oc[[1]], col='red')
> oc2 = ocontour(x,external=TRUE)
> lines(oc2[[1]])
> points(oc2[[1]], col='green')
```



See code in EBImage/src/ocontour.c

3.5 ImageMagick filters

These are not really useful functions, but can be nice when experimenting with Image Processing.

```
> x = readImage(system.file("images", "lena.gif", package="EBImage"))
> if (interactive()) display(x)
> y = minFilter(x)
```

```
> if (interactive()) display(y, title='minFilter')
> y = maxFilter(x)
> if (interactive()) display(y, title='maxFilter')
> y = meanFilter(x)
> if (interactive()) display(y, title='meanFilter')
> y = medianFilter(x)
> if (interactive()) display(y, title='medianFilter')
> y = gradientFilter(x)
> if (interactive()) display(y, title='gradientFilter')
> y = nonpeakFilter(x)
> if (interactive()) display(y, title='nonpeakFilter')
> y = stdFilter(x)
> if (interactive()) display(y, title='stdFilter')
> y = modeFilter(x)
> if (interactive()) display(y, title='modeFilter')
```

min, max, mean, median, gradient, nonpeak, std and mode filters.

See code in EBImage/src/filters_magick.c

3.6 imageReplace & colorize

```
> x = readImage(system.file("images", "lena.gif", package="EBImage"))
> if (interactive()) display(colorize(x))
```