Errata

for

Burger, Burge: Principles of Digital Image Processing – Fundamental Techniques

www.imagingbook.com

as follows:

$$\begin{aligned} &D_0 \leftarrow I * H_0^S \quad D_1 \leftarrow I * H_1^S \quad D_2 \leftarrow I * H_2^S \quad D_3 \leftarrow I * H_3^S \\ &D_4 \leftarrow -D_0 \quad D_5 \leftarrow -D_1 \quad D_6 \leftarrow -D_2 \quad D_7 \leftarrow -D_3. \end{aligned} \tag{6.22}$$

The edge strength E^S at position (u, v) is defined as the maximum of the eight filter outputs; i. e.,

$$\begin{aligned} E^S(u,v) &\triangleq \max\big(D_0(u,v), D_1(u,v), D_2(u,v), D_3(u,v), D_4(u,v), \dots D_7(u,v)\big) \\ &= \max\big(|D_0(u,v)|, |D_1(u,v)|, |D_2(u,v)|, |D_3(u,v)|\big), \end{aligned} \tag{6.23}$$

and the strongest-responding filter also determines the local edge orientation as

$$\Phi^S(u,v) \triangleq \frac{\pi}{4} j\,, \qquad \text{with } \; j = \operatorname*{argmax}_{0 \le i \le 7} D_i(u,v). \tag{6.24}$$

Another classic compass operator is the one proposed by *Kirsch* [27], which is also based on eight directional filters with the following kernels:

$$H_0^K = \begin{bmatrix} -5 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & 3 & 3 \end{bmatrix} \qquad H_4^K = \begin{bmatrix} 3 & 3 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & -5 \end{bmatrix}, \tag{6.25}$$

$$H_1^K = \begin{bmatrix} -5 & -5 & 3 \\ -5 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix} \qquad H_5^K = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & -5 \\ 3 & -5 & -5 \end{bmatrix}, \tag{6.26}$$

$$H_2^K = \begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix} \qquad H_6^K = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix}, \tag{6.27}$$

$$H_3^K = \begin{bmatrix} 3 & -5 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & 3 \end{bmatrix} \qquad H_7^K = \begin{bmatrix} 3 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & -5 & 3 \end{bmatrix}. \tag{6.28}$$

Again, because of the symmetries, only four of the eight filters need to be applied and the results may be combined in the same way as described above for the extended Sobel operator. In practice, this and other "compass operators" show only minor benefits over the simpler operators described earlier, including the small advantage of not requiring the computation of square roots (which is considered a relatively "expensive" operation).

```
1  // File Index_To_Rgb.java
2
3  import ij.ImagePlus;
4  import ij.plugin.filter.PlugInFilter;
5  import ij.process.ColorProcessor;
6  import ij.process.ImageProcessor;
7  import java.awt.image.IndexColorModel;
8
9  public class Index_To_Rgb implements PlugInFilter {
10   static final int R = 0, G = 1, B = 2;
11
12   public int setup(String arg, ImagePlus im) {
13     return DOES_8C + NO_CHANGES; //does not alter original image
14   }
15
16   public void run(ImageProcessor ip) {
17     int w = ip.getWidth();
18     int h = ip.getHeight();
19
20     //retrieve the color table (palette) for R,G,B
21     IndexColorModel icm =
22         (IndexColorModel) ip.getColorModel();
23     int mapSize = icm.getMapSize();
24     byte[] Rmap = new byte[mapSize]; icm.getReds(Rmap);
25     byte[] Gmap = new byte[mapSize]; icm.getGreens(Gmap);
26     byte[] Bmap = new byte[mapSize]; icm.getBlues(Bmap);
27
28     //create new 24-bit RGB image
29     ColorProcessor cp = new ColorProcessor(w,h);
30     int[] RGB = new int[3];
31     for (int v = 0; v < h; v++) {
32       for (int u = 0; u < w; u++) {
33         int idx = ip.getPixel(u, v);
34         RGB[R] = 0xFF & Rmap[idx]; // treat maps as
35         RGB[G] = 0xFF & Gmap[idx]; // UNSIGNED byte!
36         RGB[B] = 0xFF & Bmap[idx];
37         cp.putPixel(u, v, RGB);    // putPixel() instead of set()
38       }
39     }
40     ImagePlus cimg = new ImagePlus("RGB Image",cp);
41     cimg.show();
42   }
43
44 } // end of class Index_To_Rgb
```

Program 8.4 Converting an indexed image to a true color RGB image (ImageJ plugin).

```
1    int tIdx = 2; // index of transparent color
2    IndexColorModel icm2 = new
3      IndexColorModel(pixBits, mapSize, Rmap, Gmap, Bmap, tIdx);
4    ip.setColorModel(icm2);
```