A
Mathematical Notation

A.1 Symbols

The following symbols are used in the main text primarily with the denotations given below. While some symbols may be used for purposes other than the ones listed, the meaning should always be clear in the particular context.

$\{a, b, c, d, \ldots\}$ — A *set*; i. e., an unordered collection of distinct elements. A particular element x can be contained in a set at most once. A set may also be empty ({ }).

$(a_1, a_2, \ldots a_n)$ — A *vector*; i. e., a fixed-size collection of elements of the same type. $(a_1, a_2, \ldots a_n)^T$ denotes the *transposed* (i. e., column) vector. In programming, vectors are usually implemented as one-dimensional arrays, with elements being referred to by position (index).

$[c_1, c_2, \ldots c_m]$ — A *sequence* or *list*; i. e., a collection of elements of variable length. Elements can be added to the sequence (inserted) or deleted from the sequence. A sequence may be empty ([ ]). In programming, sequences are usually implemented with dynamic data structures, such as linked lists. Java's *Collections* framework (see also Vol. 1 [14, Appendix B.2.7]) provides numerous ready-to-use implementations.

$\langle \alpha_1, \alpha_2, \ldots \alpha_k \rangle$	A *tuple*; i. e., an ordered list of elements, each possibly of a different type. Tuples are typically implemented as *objects* (in Java or C++) or *structures* (in C) with elements being referred to by name.
$\neg$	Logical "not" operator.
$\wedge$	Logical "and" operator.
$*$	Linear convolution operator.
$\circledast$	Linear correlation operator (Sec. 11.1.1).
$\oplus$	Morphological dilation operator (see Vol. 1 [14, Sec. 7.2.3]).
$\ominus$	Morphological erosion operator (see Vol. 1 [14, Sec. 7.2.4]).
∂	Partial derivative operator (see Vol. 1 [14, Sec. 6.2.1]). For example, $\frac{\partial f}{\partial x}(x, y)$ denotes the *first* derivative of the function $f(x, y)$ along the x variable at position (x, y), $\frac{\partial^2 f}{\partial^2 x}(x, y)$ is the *second* derivative, etc.
∇	Gradient. ∇f is the vector of partial derivatives of a multidimensional function f (see Vol. 1 [14, Sec. 6.2.1]).
$\lfloor x \rfloor$	"Floor" of x, the largest integer $z \in \mathbb{Z}$ smaller than $x \in \mathbb{R}$ (i. e., $z = \lfloor x \rfloor \leq x$). For example, $\lfloor 3.141 \rfloor = 3$, $\lfloor -1.2 \rfloor = -2$.
a	Pixel value (usually $0 \leq a < K$).
$\mathrm{Arctan}(y, x)$	Inverse tangent function, similar to $\arctan\left(\frac{y}{x}\right) = \tan^{-1}\left(\frac{y}{x}\right)$ but with two arguments and returning angles in the range $[-\pi, +\pi]$ (i. e., covering all four quadrants). It corresponds to the Java method `Math.atan2(y,x)`.
$\mathrm{card}\{\ldots\}$	Cardinality (size) of a set, $\mathrm{card}\, A \equiv \lvert A \rvert$.
DFT	Discrete Fourier transform (Sec. 7.3).
$\mathcal{F}$	Continuous Fourier transform (Sec. 7.1.4).
$g(x)$, $g(x, y)$	One- and two-dimensional *continuous* functions ($x, y \in \mathbb{R}$).
$g(u)$, $g(u, v)$	One- and two-dimensional *discrete* functions ($u, v \in \mathbb{Z}$).
$G(m)$, $G(m, n)$	One- and two-dimensional discrete Fourier spectra ($m, n \in \mathbb{Z}$).

$\mathsf{h}(i)$	Histogram of an image at pixel value (or bin) i (see Vol. 1 [14, Sec. 3.1]).
$\mathsf{H}(i)$	Cumulative histogram of an image at pixel value (or bin) i (see Vol. 1 [14, Sec. 3.6]).
$I(u, v)$	Intensity value of the image I at (integer) position (u, v).
i	Imaginary unit, $\mathrm{i}^2 = -1$ (see Sec. A.3).
K	Number of possible pixel values.
M, N	Number of columns (width) and rows (height) of an image ($0 \le u < M$, $0 \le v < N$).
mod	Modulus operator: $(a \bmod b)$ is the remainder of the integer division a/b.
$\mathsf{p}(i)$	Probability density function (see Vol. 1 [14, Sec. 4.6.1]).
$\mathsf{P}(i)$	Probability distribution function or cumulative probability density (see Vol. 1 [14, Sec. 4.6.1]).
$\mathcal{Q}$	Quadrilateral (Sec. 10.1.4).
$\mathrm{round}(x)$	Rounding function: rounds x to the nearest integer. $\mathrm{round}(x) = \lfloor x + 0.5 \rfloor$ (Sec. 10.3.1).
$\mathrm{truncate}(x)$	Truncation function: truncates x toward zero to the closest integer. For example, $\mathrm{truncate}(3.141) = 3$, $\mathrm{truncate}(-2.5) = -2$.
$\mathcal{S}_1$	Unit square (Sec. 10.1.4).

A.2 Set Operators

$\lvert A \rvert$	The size (number of elements) of the set A (equivalent to $\mathrm{card}\, A$).
$\forall_x \ldots$	"All" quantifier (for all x, ...).
$\exists_x \ldots$	"Exists" quantifier (there is some x for which ...).
$\cup$	Set union (e. g., $A \cup B$).
$\cap$	Set intersection (e. g., $A \cap B$).
$\bigcup_{\mathcal{R}_i}$	Union over multiple sets $\mathcal{R}_i$.

$\bigcap_{\mathcal{R}_i}$ Intersection over multiple sets $\mathcal{R}_i$.

A.3 Complex Numbers

Definitions:

$$z = a + \mathrm{i}b, \qquad z, \mathrm{i} \in \mathbb{C},\ a, b \in \mathbb{R},\ i^2 = -1, \tag{A.1}$$

$$z^* = a - \mathrm{i}b \qquad \text{(conjugate complex)}, \tag{A.2}$$

$$sz = sa + \mathrm{i}sb, \qquad s \in \mathbb{R}, \tag{A.3}$$

$$|z| = \sqrt{a^2 + b^2}, \quad |sz| = s|z|, \tag{A.4}$$

$$\begin{aligned} z &= a + \mathrm{i}b \\ &= |z| \cdot (\cos\psi + \mathrm{i}\sin\psi) && \text{(A.5)} \\ &= |z| \cdot e^{\mathrm{i}\psi}, \quad \text{where } \psi = \tan^{-1}(b/a), && \text{(A.6)} \end{aligned}$$

$$\mathrm{Re}\big(a + \mathrm{i}b\big) = a, \quad \mathrm{Re}\big(e^{\mathrm{i}\varphi}\big) = \cos\varphi, \tag{A.7}$$

$$\mathrm{Im}\big(a + \mathrm{i}b\big) = b, \quad \mathrm{Im}\big(e^{\mathrm{i}\varphi}\big) = \sin\varphi, \tag{A.8}$$

$$e^{\mathrm{i}\varphi} = \cos\varphi + \mathrm{i} \cdot \sin\varphi, \tag{A.9}$$

$$e^{-\mathrm{i}\varphi} = \cos\varphi - \mathrm{i} \cdot \sin\varphi, \tag{A.10}$$

$$\cos(\varphi) = \tfrac{1}{2} \cdot \big(e^{\mathrm{i}\varphi} + e^{-\mathrm{i}\varphi}\big), \tag{A.11}$$

$$\sin(\varphi) = \tfrac{1}{2i} \cdot \big(e^{\mathrm{i}\varphi} - e^{-\mathrm{i}\varphi}\big), \tag{A.12}$$

Arithmetic operations:

$$\begin{aligned} z_1 &= (a_1 + \mathrm{i}b_1) = |z_1|\, e^{\mathrm{i}\varphi_1}, \\ z_2 &= (a_2 + \mathrm{i}b_2) = |z_2|\, e^{\mathrm{i}\varphi_2}, \end{aligned}$$

$$z_1 + z_2 = (a_1 + a_2) + \mathrm{i}(b_1 + b_2), \tag{A.13}$$

$$\begin{aligned} z_1 \cdot z_2 &= (a_1 a_2 - b_1 b_2) + \mathrm{i}(a_1 b_2 + b_1 a_2) && \text{(A.14)} \\ &= |z_1| \cdot |z_2| \cdot e^{\mathrm{i}(\varphi_1 + \varphi_2)}, && \text{(A.15)} \end{aligned}$$

$$\begin{aligned} \frac{z_1}{z_2} &= \frac{a_1 a_2 + b_1 b_2}{a_2^2 + b_2^2} + \mathrm{i}\,\frac{a_2 b_1 - a_1 b_2}{a_2^2 + b_2^2} && \text{(A.16)} \\ &= \frac{|z_1|}{|z_2|} \cdot e^{\mathrm{i}(\varphi_1 - \varphi_2)}. && \text{(A.17)} \end{aligned}$$

B
Source Code

B.1 Combined Region Labeling and Contour Tracing

The following Java source code represents a complete implementation of the combined region labeling and contour tracing algorithm described in Sec. 2.2. It consists of the following classes (files):

- `Contour_Tracing_Plugin`: a sample ImageJ plugin that demonstrates the use of this region labeling implementation.
- `Contour` (p. 285): a class representing a contour object.
- `BinaryRegion` (p. 286): a class representing a binary region object.
- `ContourTracer` (p. 287): the actual region labeler and contour tracer. This class is instantiated to create a region labeler for a given image.
- `ContourOverlay` (p. 292): a class for displaying contours as vector graphics on top of images.

B.1.1 `Contour_Tracing_Plugin` (Class)

```
1 import java.util.List;
2 import regions.BinaryRegion;
3 import regions.RegionLabeling;
4 import contours.Contour;
5 import contours.ContourOverlay;
```

```
 6 import contours.ContourTracer;
 7
 8 import ij.IJ;
 9 import ij.ImagePlus;
10 import ij.gui.ImageWindow;
11 import ij.plugin.filter.PlugInFilter;
12 import ij.process.ImageProcessor;
13
14 // This plugin implements the combined contour tracing and
15 // component labeling algorithm as described in [17].
16 // It uses the ContourTracer class to create lists of points
17 // representing the internal and external contours of each region in
18 // the binary image. Instead of drawing directly into the image,
19 // we make use of ImageJ's ImageCanvas to draw the contours
20 // in a separate layer on top of the image. It illustrates how to use
21 // the Java2D API to draw the polygons and scale and transform
22 // them to match ImageJ's zooming.
23
24
25 public class Contour_Tracing_Plugin implements PlugInFilter
26 {
27   ImagePlus origImage = null;
28   String origTitle = null;
29   static boolean verbose = true;
30
31   public int setup(String arg, ImagePlus im) {
32     origImage = im;
33     origTitle = im.getTitle();
34     RegionLabeling.setVerbose(verbose);
35     return DOES_8G + NO_CHANGES;
36   }
37
38   public void run(ImageProcessor ip) {
39     ImageProcessor ip2 = ip.duplicate();
40
41     // label regions and trace contours
42     ContourTracer tracer = new ContourTracer(ip2);
43
44     // extract contours and regions
45     List<Contour> outerContours = tracer.getOuterContours();
46     List<Contour> innerContours = tracer.getInnerContours();
47     List<BinaryRegion> regions = tracer.getRegions();
48     if (verbose) printRegions(regions);
49
50     // change lookup table to show gray regions
51     ip2.setMinAndMax(0,512);
52     // create an image with overlay to show the contours
53     ImagePlus im2 = new ImagePlus("Contours of " + origTitle, ip2);
54     ContourOverlay cc = new ContourOverlay(im2, outerContours,
           innerContours);
55     new ImageWindow(im2, cc);
56   }
57
```

```
58   void printRegions(List<BinaryRegion> regions) {
59     for (BinaryRegion r: regions) {
60       IJ.write("" + r);
61     }
62   }
63
64 } // end of class Contour_Tracing_Plugin
```

B.1.2 `Contour` (Class)

```
 1 package contours;
 2 import ij.IJ;
 3 import java.awt.Point;
 4 import java.awt.Polygon;
 5 import java.awt.Shape;
 6 import java.awt.geom.Ellipse2D;
 7 import java.util.ArrayList;
 8 import java.util.Iterator;
 9 import java.util.List;
10
11 public class Contour {
12   static int INITIAL_SIZE = 50;
13   int label;
14   List<Point> points;
15
16   Contour (int label, int size) {
17     this.label = label;
18     points = new ArrayList<Point>(size);
19   }
20
21   Contour (int label) {
22     this.label = label;
23     points = new ArrayList<Point>(INITIAL_SIZE);
24   }
25
26   void addPoint (Point n) {
27     points.add(n);
28   }
29
30   Shape makePolygon() {
31     int m = points.size();
32     if (m>1) {
33       int[] xPoints = new int[m];
34       int[] yPoints = new int[m];
35       int k = 0;
36       Iterator<Point> itr = points.iterator();
37       while (itr.hasNext() && k < m) {
38         Point cpt = itr.next();
39         xPoints[k] = cpt.x;
40         yPoints[k] = cpt.y;
41         k = k + 1;
42       }
```

```
43        return new Polygon(xPoints, yPoints, m);
44      }
45      else { // use circles for isolated pixels
46        Point cpt = points.get(0);
47        return new Ellipse2D.Double
48            (cpt.x-0.1, cpt.y-0.1, 0.2, 0.2);
49      }
50    }
51
52    static Shape[] makePolygons(List<Contour> contours) {
53      if (contours == null)
54        return null;
55      else {
56        Shape[] pa = new Shape[contours.size()];
57        int i = 0;
58        for (Contour c: contours) {
59          pa[i] = c.makePolygon();
60          i = i + 1;
61        }
62        return pa;
63      }
64    }
65
66    void moveBy (int dx, int dy) {
67      for (Point pt: points) {
68        pt.translate(dx,dy);
69      }
70    }
71
72    static void moveContoursBy
73             (List<Contour> contours, int dx, int dy) {
74      for (Contour c: contours) {
75        c.moveBy(dx, dy);
76      }
77    }
78
79  } // end of class Contour
```

B.1.3 BinaryRegion (Class)

```
1  package regions;
2  import java.awt.Rectangle;
3  import java.awt.geom.Point2D;
4
5  public class BinaryRegion {
6    int label;
7    int numberOfPixels = 0;
8    double xc = Double.NaN;
9    double yc = Double.NaN;
10   int left = Integer.MAX_VALUE;
11   int right = -1;
12   int top = Integer.MAX_VALUE;
```

```
13   int bottom = -1;
14
15   int x_sum = 0;
16   int y_sum = 0;
17   int x2_sum = 0;
18   int y2_sum = 0;
19
20   public BinaryRegion(int id){
21     this.label = id;
22   }
23
24   public int getSize() {
25     return this.numberOfPixels;
26   }
27
28   public Rectangle getBoundingBox() {
29     if (left == Integer.MAX_VALUE)
30       return null;
31     else
32       return new Rectangle
33         (left, top, right-left+1, bottom-top+1);
34   }
35
36   public Point2D.Double getCenter(){
37     if (Double.isNaN(xc))
38       return null;
39     else
40       return new Point2D.Double(xc, yc);
41   }
42
43   public void addPixel(int x, int y){
44     numberOfPixels = numberOfPixels + 1;
45     x_sum = x_sum + x;
46     y_sum = y_sum + y;
47     x2_sum = x2_sum + x*x;
48     y2_sum = y2_sum + y*y;
49     if (x<left) left = x;
50     if (y<top) top = y;
51     if (x>right) right = x;
52     if (y>bottom) bottom = y;
53   }
54
55   public void update(){
56     if (numberOfPixels > 0){
57       xc = x_sum / numberOfPixels;
58       yc = y_sum / numberOfPixels;
59     }
60   }
61
62 } // end of class BinaryRegion
```

B.1.4 ContourTracer (Class)

```
1  package contours;
2  import java.awt.Point;
3  import java.util.ArrayList;
4  import java.util.LinkedList;
5  import java.util.List;
6  import regions.BinaryRegion;
7  import ij.IJ;
8  import ij.process.ImageProcessor;
9
10 public class ContourTracer {
11   static final byte FOREGROUND = 1;
12   static final byte BACKGROUND = 0;
13   static boolean beVerbose = true;
14
15   List<Contour> outerContours = null;
16   List<Contour> innerContours = null;
17   List<BinaryRegion> allRegions = null;
18   int regionId = 0;
19
20   ImageProcessor ip = null;
21   int width;
22   int height;
23   byte[][] pixelArray;
24   int[][] labelArray;
25
26   // label values in labelArray can be:
27   //   0 ... unlabeled
28   // -1 ... previously visited background pixel
29   // > 0 ... a valid label
30
31   // constructor method
32   public ContourTracer (ImageProcessor ip) {
33     this.ip = ip;
34     this.width = ip.getWidth();
35     this.height = ip.getHeight();
36     makeAuxArrays();
37     findAllContours();
38     collectRegions();
39   }
40
41   public static void setVerbose(boolean verbose) {
42     beVerbose = verbose;
43   }
44
45   public List<Contour> getOuterContours() {
46     return outerContours;
47   }
48
49   public List<Contour> getInnerContours() {
50     return innerContours;
51   }
52
53   public List<BinaryRegion> getRegions() {
```

```
54    return allRegions;
55  }
56
57  // nonpublic methods
58
59  void makeAuxArrays() {
60    int h = ip.getHeight();
61    int w = ip.getWidth();
62    pixelArray = new byte[h+2][w+2];
63    labelArray = new int[h+2][w+2];
64    // initialize auxiliary arrays
65    for (int v = 0; v < h+2; v++) {
66      for (int u = 0; u < w+2; u++) {
67        if (ip.get(u-1,v-1) == 0)
68          pixelArray[v][u] = BACKGROUND;
69        else
70          pixelArray[v][u] = FOREGROUND;
71      }
72    }
73  }
74
75  Contour traceOuterContour (int cx, int cy, int label) {
76    Contour cont = new Contour(label);
77    traceContour(cx, cy, label, 0, cont);
78    return cont;
79  }
80
81  Contour traceInnerContour(int cx, int cy, int label) {
82    Contour cont = new Contour(label);
83    traceContour(cx, cy, label, 1, cont);
84    return cont;
85  }
86
87  // trace one contour starting at (xS,yS) in direction dS
88  Contour traceContour (int xS, int yS, int label, int dS, Contour
        cont) {
89    int xT, yT; // T = successor of starting point (xS,yS)
90    int xP, yP; // P = previous contour point
91    int xC, yC; // C = current contour point
92    Point pt = new Point(xS, yS);
93    int dNext = findNextPoint(pt, dS);
94    cont.addPoint(pt);
95    xP = xS; yP = yS;
96    xC = xT = pt.x;
97    yC = yT = pt.y;
98
99    boolean done = (xS==xT && yS==yT); // true if isolated pixel
100
101   while (!done) {
102     labelArray[yC][xC] = label;
103     pt = new Point(xC, yC);
104     int dSearch = (dNext + 6) % 8;
105     dNext = findNextPoint(pt, dSearch);
```

```
106       xP = xC; yP = yC;
107       xC = pt.x; yC = pt.y;
108       // are we back at the starting position?
109       done = (xP==xS && yP==yS && xC==xT && yC==yT);
110       if (!done) {
111         cont.addPoint(pt);
112       }
113     }
114     return cont;
115   }
116
117   int findNextPoint (Point pt, int dir) {
118     // starts at Point pt in direction dir, returns the
119     // final tracing direction, and modifies pt
120     final int[][] delta = {
121       { 1,0}, { 1, 1}, {0, 1}, {-1, 1},
122       {-1,0}, {-1,-1}, {0,-1}, { 1,-1}};
123     for (int i = 0; i < 7; i++) {
124       int x = pt.x + delta[dir][0];
125       int y = pt.y + delta[dir][1];
126       if (pixelArray[y][x] == BACKGROUND) {
127         // mark surrounding background pixels
128         labelArray[y][x] = -1;
129         dir = (dir + 1) % 8;
130       }
131       else { // found a nonbackground pixel
132         pt.x = x; pt.y = y;
133         break;
134       }
135     }
136     return dir;
137   }
138
139   void findAllContours() {
140     outerContours = new ArrayList<Contour>(50);
141     innerContours = new ArrayList<Contour>(50);
142     int label = 0;   // current label
143
144     // scan top to bottom, left to right
145     for (int v = 1; v < pixelArray.length-1; v++) {
146       label = 0; // no label
147       for (int u = 1; u < pixelArray[v].length-1; u++) {
148
149         if (pixelArray[v][u] == FOREGROUND) {
150           if (label != 0) { // keep using the same label
151             labelArray[v][u] = label;
152           }
153           else {
154             label = labelArray[v][u];
155             if (label == 0) {
156               // unlabeled—new outer contour
157               regionId = regionId + 1;
158               label = regionId;
```

```
159             Contour oc = traceOuterContour(u, v, label);
160             outerContours.add(oc);
161             labelArray[v][u] = label;
162           }
163         }
164       }
165       else { // background pixel
166         if (label != 0) {
167           if (labelArray[v][u] == 0) {
168             // unlabeled—new inner contour
169             Contour ic = traceInnerContour(u-1, v, label);
170             innerContours.add(ic);
171           }
172           label = 0;
173         }
174       }
175     }
176   }
177   // shift back to original coordinates
178   Contour.moveContoursBy (outerContours, -1, -1);
179   Contour.moveContoursBy (innerContours, -1, -1);
180 }
181
182
183 // creates a container of BinaryRegion objects
184 // collects the region pixels from the label image
185 // and computes the statistics for each region
186 void collectRegions() {
187   int maxLabel = this.regionId;
188   int startLabel = 1;
189   BinaryRegion[] regionArray =
190         new BinaryRegion[maxLabel + 1];
191   for (int i = startLabel; i <= maxLabel; i++) {
192     regionArray[i] = new BinaryRegion(i);
193   }
194   for (int v = 0; v < height; v++) {
195     for (int u = 0; u < width; u++) {
196       int lb = labelArray[v][u];
197       if (lb >= startLabel && lb <= maxLabel
198           && regionArray[lb]!=null) {
199         regionArray[lb].addPixel(u, v);
200       }
201     }
202   }
203
204   // create a list of regions to return, collect nonempty regions
205   List<BinaryRegion> regionList =
206             new LinkedList<BinaryRegion>();
207     for (BinaryRegion r: regionArray) {
208       if (r != null && r.getSize()>0) {
209         r.update(); // compute the statistics for this region
210         regionList.add(r);
211       }
```

```
212       }
213       allRegions = regionList;
214     }
215
216 } // end of class ContourTracer
```

B.1.5 `ContourOverlay` (Class)

```
 1 package contours;
 2 import ij.ImagePlus;
 3 import ij.gui.ImageCanvas;
 4 import java.awt.BasicStroke;
 5 import java.awt.Color;
 6 import java.awt.Graphics;
 7 import java.awt.Graphics2D;
 8 import java.awt.Polygon;
 9 import java.awt.RenderingHints;
10 import java.awt.Shape;
11 import java.awt.Stroke;
12 import java.util.List;
13
14 public class ContourOverlay extends ImageCanvas {
15   private static final long serialVersionUID = 1L;
16   static float strokeWidth = 0.5f;
17   static int capsstyle = BasicStroke.CAP_ROUND;
18   static int joinstyle = BasicStroke.JOIN_ROUND;
19   static Color outerColor = Color.black;
20   static Color innerColor = Color.white;
21   static float[] outerDashing = {strokeWidth * 2.0f, strokeWidth *
         2.5f};
22   static float[] innerDashing = {strokeWidth * 0.5f, strokeWidth *
         2.5f};
23   static boolean DRAW_CONTOURS = true;
24
25   Shape[] outerContourShapes = null;
26   Shape[] innerContourShapes = null;
27
28   public ContourOverlay(ImagePlus im,
29         List<Contour> outerCs, List<Contour> innerCs)
30   {
31     super(im);
32     if (outerCs != null)
33       outerContourShapes = Contour.makePolygons(outerCs);
34     if (innerCs != null)
35       innerContourShapes = Contour.makePolygons(innerCs);
36   }
37
38   public void paint(Graphics g) {
39     super.paint(g);
40     drawContours(g);
41   }
42
```

```
43   // nonpublic methods
44
45   private void drawContours(Graphics g) {
46     Graphics2D g2d = (Graphics2D) g;
47     g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
           RenderingHints.VALUE_ANTIALIAS_ON);
48
49     // scale and move overlay to the pixel centers
50     double mag = this.getMagnification();
51     g2d.scale(mag, mag);
52     g2d.translate(0.5-this.srcRect.x, 0.5-this.srcRect.y);
53
54     if (DRAW_CONTOURS) {
55       Stroke solidStroke = new BasicStroke
56           (strokeWidth, capsstyle, joinstyle);
57       Stroke dashedStrokeOuter = new BasicStroke
58           (strokeWidth, capsstyle, joinstyle, 1.0f, outerDashing, 0.0
                f);
59       Stroke dashedStrokeInner = new BasicStroke
60           (strokeWidth, capsstyle, joinstyle, 1.0f, innerDashing, 0.0
                f);
61
62       if (outerContourShapes != null)
63         drawShapes(outerContourShapes, g2d, solidStroke,
               dashedStrokeOuter, outerColor);
64       if (innerContourShapes != null)
65         drawShapes(innerContourShapes, g2d, solidStroke,
               dashedStrokeInner, innerColor);
66     }
67   }
68
69   void drawShapes(Shape[] shapes, Graphics2D g2d,
70        Stroke solidStrk, Stroke dashedStrk, Color col) {
71     g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
           RenderingHints.VALUE_ANTIALIAS_ON);
72     g2d.setColor(col);
73     for (int i = 0; i < shapes.length; i++) {
74       Shape s = shapes[i];
75       if (s instanceof Polygon)
76         g2d.setStroke(dashedStrk);
77       else
78         g2d.setStroke(solidStrk);
79       g2d.draw(s);
80     }
81   }
82
83 } // end of class ContourOverlay
```

B.2 Harris Corner Detector

The following Java source code represents a complete implementation of the Harris corner detector, as described in Ch. 4. It consists of the following classes (files):

- `Harris_Corner_Plugin`: a sample ImageJ plugin that demonstrates the use of the corner detector.
- `Corner` (p. 295): a class representing an individual corner object.
- `HarrisCornerDetector` (p. 296): the actual corner detector. This class is instantiated to create a corner detector for a given image.

B.2.1 `Harris_Corner_Plugin` (Class)

```
1  import harris.HarrisCornerDetector;
2  import ij.IJ;
3  import ij.ImagePlus;
4  import ij.gui.GenericDialog;
5  import ij.plugin.filter.PlugInFilter;
6  import ij.process.ImageProcessor;
7
8  public class Harris_Corner_Plugin implements PlugInFilter {
9      ImagePlus im;
10     static float alpha = HarrisCornerDetector.DEFAULT_ALPHA;
11     static int threshold = HarrisCornerDetector.DEFAULT_THRESHOLD;
12     static int nmax = 0; //points to show
13
14     public int setup(String arg, ImagePlus im) {
15       this.im = im;
16         if (arg.equals("about")) {
17             showAbout();
18             return DONE;
19         }
20         return DOES_8G + NO_CHANGES;
21     }
22
23     public void run(ImageProcessor ip) {
24         if (!showDialog()) return; //dialog canceled or error
25         HarrisCornerDetector hcd =
26             new HarrisCornerDetector(ip,alpha,threshold);
27         hcd.findCorners();
28         ImageProcessor result = hcd.showCornerPoints(ip);
29         ImagePlus win =
30             new ImagePlus("Corners from " + im.getTitle(),result);
31         win.show();
32     }
33
34     void showAbout() {
35         String cn = getClass().getName();
```

```
36        IJ.showMessage("About "+cn+" ...",
37            "Harris Corner Detector");
38    }
39
40    private boolean showDialog() {
41      // display dialog, and return false if canceled or in error.
42      GenericDialog dlg = new GenericDialog("Harris Corner Detector",
            IJ.getInstance());
43      float def_alpha = HarrisCornerDetector.DEFAULT_ALPHA;
44      dlg.addNumericField("Alpha (default: "+def_alpha+")", alpha, 3)
            ;
45      int def_threshold = HarrisCornerDetector.DEFAULT_THRESHOLD;
46      dlg.addNumericField("Threshold (default: "+def_threshold+")",
            threshold, 0);
47      dlg.addNumericField("Max. points (0 = show all)", nmax, 0);
48      dlg.showDialog();
49      if(dlg.wasCanceled())
50        return false;
51      if(dlg.invalidNumber()) {
52        IJ.showMessage("Error", "Invalid input number");
53        return false;
54      }
55      alpha = (float) dlg.getNextNumber();
56      threshold = (int) dlg.getNextNumber();
57      nmax = (int) dlg.getNextNumber();
58      return true;
59    }
60 } // end of class Harris_Corner_Plugin
```

B.2.2 File Corner (Class)

```
1  package harris;
2  import ij.process.ImageProcessor;
3
4  class Corner implements Comparable {
5    int u;
6    int v;
7    float q;
8
9    Corner (int u, int v, float q) {
10     this.u = u;
11     this.v = v;
12     this.q = q;
13   }
14
15   public int compareTo (Object obj) {
16     // used for sorting corners by corner strength q
17     Corner c2 = (Corner) obj;
18     if (this.q > c2.q) return -1;
19     if (this.q < c2.q) return 1;
20     else return 0;
21   }
```

```
22
23   double dist2 (Corner c2) {
24     // returns the squared distance between this corner and corner c2
25     int dx = this.u - c2.u;
26     int dy = this.v - c2.v;
27     return (dx*dx)+(dy*dy);
28   }
29
30   void draw(ImageProcessor ip) {
31     // draw this corner as a black cross in ip
32     int paintvalue = 0; // black
33     int size = 2;
34     ip.setValue(paintvalue);
35     ip.drawLine(u-size,v,u+size,v);
36     ip.drawLine(u,v-size,u,v+size);
37   }
38
39 } // end of class Corner
```

B.2.3 File HarrisCornerDetector (Class)

```
1  package harris;
2  import ij.IJ;
3  import ij.ImagePlus;
4  import ij.plugin.filter.Convolver;
5  import ij.process.Blitter;
6  import ij.process.ByteProcessor;
7  import ij.process.FloatProcessor;
8  import ij.process.ImageProcessor;
9  import java.util.Arrays;
10 import java.util.Collections;
11 import java.util.List;
12 import java.util.Vector;
13
14 public class HarrisCornerDetector {
15
16   public static final float DEFAULT_ALPHA = 0.050f;
17   public static final int DEFAULT_THRESHOLD = 20000;
18   float alpha = DEFAULT_ALPHA;
19   int threshold = DEFAULT_THRESHOLD;
20   double dmin = 10;
21   final int border = 20;
22
23   // filter kernels (1D part of separable 2D filters)
24   final float[] pfilt = {0.223755f,0.552490f,0.223755f};
25   final float[] dfilt = {0.453014f,0.0f,-0.453014f};
26   final float[] bfilt = {0.01563f,0.09375f,0.234375f,0.3125f
         ,0.234375f,0.09375f,0.01563f};
27               // = [1, 6, 15, 20, 15, 6, 1]/64
28   ImageProcessor ipOrig;
29   FloatProcessor A;
30   FloatProcessor B;
```

```
31  FloatProcessor C;
32  FloatProcessor Q;
33  List<Corner> corners;
34
35  HarrisCornerDetector(ImageProcessor ip) {
36    this.ipOrig = ip;
37  }
38
39  public HarrisCornerDetector(ImageProcessor ip,
40         float alpha, int threshold)
41  {
42    this.ipOrig = ip;
43    this.alpha = alpha;
44    this.threshold = threshold;
45  }
46
47  public void findCorners() {
48    makeDerivatives();
49    makeCrf(); //corner response function (CRF)
50    corners = collectCorners(border);
51    corners = cleanupCorners(corners);
52  }
53
54  void makeDerivatives() {
55    FloatProcessor Ix =
56           (FloatProcessor) ipOrig.convertToFloat();
57    FloatProcessor Iy =
58           (FloatProcessor) ipOrig.convertToFloat();
59
60    Ix = convolve1h(convolve1h(Ix,pfilt),dfilt);
61    Iy = convolve1v(convolve1v(Iy,pfilt),dfilt);
62
63    A = sqr((FloatProcessor) Ix.duplicate());
64    A = convolve2(A,bfilt);
65
66    B = sqr((FloatProcessor) Iy.duplicate());
67    B = convolve2(B,bfilt);
68
69    C = mult((FloatProcessor)Ix.duplicate(),Iy);
70    C = convolve2(C,bfilt);
71  }
72
73  void makeCrf() { // corner response function (CRF)
74    int w = ipOrig.getWidth();
75    int h = ipOrig.getHeight();
76    Q = new FloatProcessor(w,h);
77    float[] Apix = (float[]) A.getPixels();
78    float[] Bpix = (float[]) B.getPixels();
79    float[] Cpix = (float[]) C.getPixels();
80    float[] Qpix = (float[]) Q.getPixels();
81    for (int v=0; v<h; v++) {
82      for (int u=0; u<w; u++) {
83        int i = v*w+u;
```

```
84        float a = Apix[i], b = Bpix[i], c = Cpix[i];
85        float det = a*b-c*c;
86        float trace = a+b;
87        Qpix[i] = det - alpha * (trace * trace);
88      }
89    }
90  }
91
92  List<Corner> collectCorners(int border) {
93    List<Corner> cornerList = new Vector<Corner>(1000);
94    int w = Q.getWidth();
95    int h = Q.getHeight();
96    float[] Qpix = (float[]) Q.getPixels();
97    for (int v=border; v<h-border; v++){
98      for (int u=border; u<w-border; u++) {
99        float q = Qpix[v*w+u];
100       if (q>threshold && isLocalMax(Q,u,v)) {
101         Corner c = new Corner(u,v,q);
102         cornerList.add(c);
103       }
104     }
105   }
106   Collections.sort(cornerList);
107   return cornerList;
108 }
109
110 List<Corner> cleanupCorners(List<Corner> corners) {
111   double dmin2 = dmin*dmin;
112   Corner[] cornerArray = new Corner[corners.size()];
113   cornerArray = corners.toArray(cornerArray);
114   List<Corner> goodCorners =
115       new Vector<Corner>(corners.size());
116   for (int i=0; i<cornerArray.length; i++){
117     if (cornerArray[i] != null){
118       Corner c1 = cornerArray[i];
119       goodCorners.add(c1);
120       // delete all remaining corners close to c
121       for (int j=i+1; j<cornerArray.length; j++){
122         if (cornerArray[j] != null){
123           Corner c2 = cornerArray[j];
124           if (c1.dist2(c2)<dmin2)
125             cornerArray[j] = null; //delete corner
126         }
127       }
128     }
129   }
130   return goodCorners;
131 }
132
133 void printCornerPoints(List<Corner> crf) {
134   int i = 0;
135   for (Corner ipt: crf){
```

```
136       IJ.write((i++) + ": " + (int)ipt.q + " " + ipt.u + " " + ipt.v)
              ;
137     }
138   }
139
140   public ImageProcessor showCornerPoints(ImageProcessor ip) {
141     ByteProcessor ipResult = (ByteProcessor)ip.duplicate();
142     // change background image contrast and brightness
143     int[] lookupTable = new int[256];
144     for (int i=0; i<256; i++){
145       lookupTable[i] = 128 + (i/2);
146     }
147     ipResult.applyTable(lookupTable);
148     // draw corners:
149     for (Corner c: corners) {
150       c.draw(ipResult);
151     }
152     return ipResult;
153   }
154
155   void showProcessor(ImageProcessor ip, String title) {
156     ImagePlus win = new ImagePlus(title,ip);
157     win.show();
158   }
159
160   // utility methods for float processors —
161
162   static FloatProcessor convolve1h
163                 (FloatProcessor p, float[] h) {
164     Convolver conv = new Convolver();
165     conv.setNormalize(false);
166     conv.convolve(p, h, 1, h.length);
167     return p;
168   }
169
170   static FloatProcessor convolve1v
171                 (FloatProcessor p, float[] h) {
172     Convolver conv = new Convolver();
173     conv.setNormalize(false);
174     conv.convolve(p, h, h.length, 1);
175     return p;
176   }
177
178   static FloatProcessor convolve2
179                 (FloatProcessor p, float[] h) {
180     convolve1h(p,h);
181     convolve1v(p,h);
182     return p;
183   }
184
185   static FloatProcessor sqr (FloatProcessor fp1) {
186     fp1.sqr();
187     return fp1;
```

```
188   }
189
190   static FloatProcessor mult (FloatProcessor fp1, FloatProcessor fp2
          ) {
191     int mode = Blitter.MULTIPLY;
192     fp1.copyBits(fp2, 0, 0, mode);
193     return fp1;
194   }
195
196   static boolean isLocalMax (FloatProcessor fp,int u,int v) {
197     int w = fp.getWidth();
198     int h = fp.getHeight();
199     if (u<=0 || u>=w-1 || v<=0 || v>=h-1)
200       return false;
201     else {
202       float[] pix = (float[]) fp.getPixels();
203       int i0 = (v-1)*w+u, i1 = v*w+u, i2 = (v+1)*w+u;
204       float cp = pix[i1];
205       return
206         cp > pix[i0-1] && cp > pix[i0] && cp > pix[i0+1] &&
207         cp > pix[i1-1] &&              cp > pix[i1+1] &&
208         cp > pix[i2-1] && cp > pix[i2] && cp > pix[i2+1] ;
209     }
210   }
211
212 } // end of class HarrisCornerDetector
```

B.3 Median-Cut Color Quantization

This is an implementation of Heckbert's median-cut color quantization algorithm [32], as described in Sec. 5.2 (Alg. 5.1–5.3). Unlike in the original algorithm, no initial uniform (scalar) quantization is used for reducing the number of image colors. Instead, all colors contained in the original image are considered in the quantization process. After the set of representative colors has been found, each image color is mapped to the closest representative in RGB color space using the Euclidean distance.

B.3.1 `ColorQuantizer` (Interface)

This is a general interface for all color quantizers.

```
1 package color;
2
3 import ij.process.ByteProcessor;
4 import ij.process.ColorProcessor;
5
6 public interface ColorQuantizer {
7   public abstract ByteProcessor quantizeImage(ColorProcessor cp);
8   public abstract int[] quantizeImage(int[] origPixels);
9   public abstract int countQuantizedColors();
10
11 }
```

B.3.2 `MedianCutQuantizer` (Class)

This class contains the main functionality of the median-cut quantizer. Figure B.1 illustrates the key data structures involved and their relationships. The classes `ColorNode` and `ColorBox` are implemented as nested classes inside `MedianCutQuantizer`. Also, notice the use of the nested enumeration class `ColorDimension` for implementing the constants `RED`, `GREEN`, and `BLUE` and the associated comparator methods.

```
1 package color;
2
3 import ij.process.ByteProcessor;
4 import ij.process.ColorProcessor;
5
6 import java.awt.image.IndexColorModel;
7 import java.util.ArrayList;
8 import java.util.Arrays;
9 import java.util.Comparator;
10 import java.util.List;
11
12 public class MedianCutQuantizer implements ColorQuantizer {
13
```

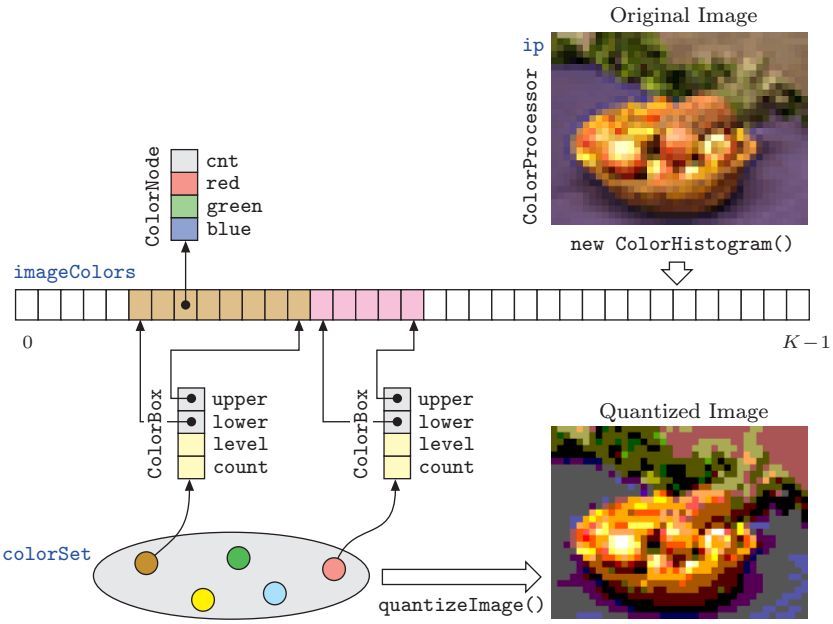

Figure B.1 Median-cut data structures. Initially, a new `ColorHistogram` is computed for the original color image (`ip` of type `ColorProcessor`). The resulting array `imageColors` of size K corresponds to the unique colors ($\mathcal{C} = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots \boldsymbol{c}_K\}$ in Alg. 5.1) contained in the original RGB image. Each cell of `imageColors` refers to a `colorNode` object ($\boldsymbol{c}_i$) that holds the associated color (`red`, `green`, `blue`) and its frequency (`cnt`) in the image. Each `colorBox` object (corresponding to a color box $\boldsymbol{b}$ in Alg. 5.1) selects a contiguous range of image colors, bounded by the indices `lower` and `upper`. The ranges of elements in `imageColors`, indexed by different `colorBox` objects, never overlap. Each element in `imageColors` is contained in exactly one `colorBox`; i. e., the color boxes held in `colorSet` ($\mathcal{B}$ in Alg. 5.1) form a partitioning of `imageColors` (`colorSet` is implemented as a list of `ColorBox` objects). To split a particular `colorBox` along a color dimension d = Red, Green, or Blue, the corresponding subrange of elements in `imageColors` is *sorted* with the property `red`, `green`, or `blue`, respectively, as the sorting key. In Java, this is quite easy to implement using the standard `Arrays.sort()` utility method and a dedicated `Comparator` object for each color dimension. Finally, the method `quantizeImage()` replaces each pixel in `ip` by the closest color in `colorSet`.

```
14  private ColorNode[] imageColors = null; // original image colors
15  private ColorNode[] quantColors = null; // quantized colors
16
17  public MedianCutQuantizer(ColorProcessor ip, int Kmax) {
18    this((int[]) ip.getPixels(), Kmax);
19  }
20
21  public MedianCutQuantizer(int[] pixels, int Kmax) {
22    quantColors = findRepresentativeColors(pixels, Kmax);
23  }
24
25  public int countQuantizedColors() {
```

```
26    return quantColors.length;
27  }
28
29  public ColorNode[] getQuantizedColors() {
30    return quantColors;
31  }
32
33  ColorNode[] findRepresentativeColors(int[] pixels, int Kmax) {
34    ColorHistogram colorHist = new ColorHistogram(pixels);
35    int K = colorHist.getNumberOfColors();
36    ColorNode[] rCols = null;
37
38    imageColors = new ColorNode[K];
39    for (int i = 0; i < K; i++) {
40      int rgb = colorHist.getColor(i);
41      int cnt = colorHist.getCount(i);
42      imageColors[i] = new ColorNode(rgb, cnt);
43    }
44
45    if (K <= Kmax) // image has fewer colors than Kmax
46      rCols = imageColors;
47    else {
48      ColorBox initialBox = new ColorBox(0, K-1, 0);
49      List<ColorBox> colorSet = new ArrayList<ColorBox>();
50      colorSet.add(initialBox);
51      int k = 1;
52      boolean done = false;
53      while (k < Kmax && !done) {
54        ColorBox nextBox = findBoxToSplit(colorSet);
55        if (nextBox != null) {
56          ColorBox newBox = nextBox.splitBox();
57          colorSet.add(newBox);
58          k = k + 1;
59        } else {
60          done = true;
61        }
62      }
63      rCols = averageColors(colorSet);
64    }
65    return rCols;
66  }
67
68  public int[] quantizeImage(int[] origPixels) {
69    int[] qantPixels = origPixels.clone();
70    for (int i = 0; i < origPixels.length; i++) {
71      ColorNode color = findClosestColor(origPixels[i]);
72      qantPixels[i] = color.rgb;
73    }
74    return qantPixels;
75  }
76
77  public ByteProcessor quantizeImage(ColorProcessor cp) {
78    if (countQuantizedColors() > 256)
```

```
79      throw new Error("cannot index to more than 256 colors");
80    int w = cp.getWidth();
81    int h = cp.getHeight();
82    int[] origPixels = (int[]) cp.getPixels();
83    byte[] idxPixels = new byte[origPixels.length];
84
85    for (int i = 0; i < origPixels.length; i++) {
86      idxPixels[i] = (byte) findClosestColorIndex(origPixels[i]);
87    }
88
89    IndexColorModel idxCm = makeIndexColorModel();
90    return new ByteProcessor(w, h, idxPixels, idxCm);
91  }
92
93  IndexColorModel makeIndexColorModel() {
94    int nColors = countQuantizedColors();
95    byte[] rMap = new byte[nColors];
96    byte[] gMap = new byte[nColors];
97    byte[] bMap = new byte[nColors];
98    for (int i=0; i<nColors; i++) {
99      rMap[i] = (byte) quantColors[i].red;
100     gMap[i] = (byte) quantColors[i].grn;
101     bMap[i] = (byte) quantColors[i].blu;
102   }
103   return new IndexColorModel(8, nColors, rMap, gMap, bMap);
104 }
105
106 ColorNode findClosestColor (int rgb) {
107   int idx = findClosestColorIndex(rgb);
108   return quantColors[idx];
109 }
110
111 int findClosestColorIndex (int rgb) {
112   int red = ((rgb & 0xFF0000) >> 16);
113   int grn = ((rgb & 0xFF00) >> 8);
114   int blu = (rgb & 0xFF);
115   int minIdx = 0;
116   int minDistance = Integer.MAX_VALUE;
117   for (int i=0; i<quantColors.length; i++) {
118     ColorNode color = quantColors[i];
119     int d2 = color.distance2(red, grn, blu);
120     if (d2 < minDistance) {
121       minDistance = d2;
122       minIdx = i;
123     }
124   }
125   return minIdx;
126 }
127
128 private ColorBox findBoxToSplit(List<ColorBox> colorBoxes) {
129   ColorBox boxToSplit = null;
130   // from the set of splitable color boxes
131   // select the one with the minimum level
```

```
132     int minLevel = Integer.MAX_VALUE;
133     for (ColorBox box : colorBoxes) {
134       if (box.colorCount() >= 2) { // box can be split
135         if (box.level < minLevel) {
136           boxToSplit = box;
137           minLevel = box.level;
138         }
139       }
140     }
141     return boxToSplit;
142   }
143
144   private ColorNode[] averageColors(List<ColorBox> colorBoxes) {
145     int n = colorBoxes.size();
146     ColorNode[] avgColors = new ColorNode[n];
147     int i = 0;
148     for (ColorBox box : colorBoxes) {
149       avgColors[i] = box.getAverageColor();
150       i = i + 1;
151     }
152     return avgColors;
153   }
154
155   // ------- class ColorNode ---------------------
156
157   class ColorNode {
158     private int rgb;
159     private int red, grn, blu;
160     private int cnt;
161
162     ColorNode (int rgb, int cnt) {
163       this.rgb = (rgb & 0xFFFFFF);
164       this.red = (rgb & 0xFF0000) >> 16;
165       this.grn = (rgb & 0xFF00) >> 8;
166       this.blu = (rgb & 0xFF);
167       this.cnt = cnt;
168     }
169
170     ColorNode (int red, int grn, int blu, int cnt) {
171       this.rgb = ((red & 0xff) << 16) | ((grn & 0xff) << 8) | blu & 0
              xff;
172       this.red = red;
173       this.grn = grn;
174       this.blu = blu;
175       this.cnt = cnt;
176     }
177
178     int distance2 (int red, int grn, int blu) {
179       // returns the squared distance between (red, grn, blu)
180       // and this color
181       int dr = this.red - red;
182       int dg = this.grn - grn;
183       int db = this.blu - blu;
```

```
184       return dr*dr + dg*dg + db*db;
185     }
186
187     public String toString() {
188       String s = this.getClass().getSimpleName();
189       s = s + " red=" + red + " green=" + grn + " blue=" + blu + "
              count=" + cnt;
190       return s;
191     }
192   }
193
194   // ——————— class ColorBox ——————————————————
195
196   class ColorBox {
197     int lower = 0;  // lower index into 'imageColors'
198     int upper = -1; // upper index into 'imageColors'
199     int level;    // split level o this color box
200     int count = 0;  // number of pixels represented by thos color box
201     int rmin, rmax; // range of contained colors in red dimension
202     int gmin, gmax; // range of contained colors in green dimension
203     int bmin, bmax; // range of contained colors in blue dimension
204
205     ColorBox(int lower, int upper, int level) {
206       this.lower = lower;
207       this.upper = upper;
208       this.level = level;
209       this.trim();
210     }
211
212     int colorCount() {
213       return upper - lower;
214     }
215
216     void trim() {
217       // recompute the boundaries of this color box
218       rmin = 255; rmax = 0;
219       gmin = 255; gmax = 0;
220       bmin = 255; bmax = 0;
221       count = 0;
222       for (int i = lower; i <= upper; i++) {
223         ColorNode color = imageColors[i];
224         count = count + color.cnt;
225         int r = color.red;
226         int g = color.grn;
227         int b = color.blu;
228         if (r > rmax) rmax = r;
229         if (r < rmin) rmin = r;
230         if (g > gmax) gmax = g;
231         if (g < gmin) gmin = g;
232         if (b > bmax) bmax = b;
233         if (b < bmin) bmin = b;
234       }
235     }
```

```
236
237     // Split this color box at the median point along its
238     // longest color dimension
239     ColorBox splitBox() {
240       if (this.colorCount() < 2) // this box cannot be split
241         return null;
242       else {
243         // find longest dimension of this box:
244         ColorDimension dim = getLongestColorDimension();
245
246         // find median along dim
247         int med = findMedian(dim);
248
249         // now split this box at the median return the resulting new
250         // box.
251         int nextLevel = level + 1;
252         ColorBox newBox = new ColorBox(med + 1, upper, nextLevel);
253         this.upper = med;
254         this.level = nextLevel;
255         this.trim();
256         return newBox;
257       }
258     }
259
260     // Find longest dimension of this color box (RED, GREEN, or BLUE)
261     ColorDimension getLongestColorDimension() {
262       int rLength = rmax - rmin;
263       int gLength = gmax - gmin;
264       int bLength = bmax - bmin;
265       if (bLength >= rLength && bLength >= gLength)
266         return ColorDimension.BLUE;
267       else if (gLength >= rLength && gLength >= bLength)
268         return ColorDimension.GREEN;
269       else return ColorDimension.RED;
270     }
271
272     // Find the position of the median in RGB space along
273     // the red, green or blue dimension, respectively.
274     int findMedian(ColorDimension dim) {
275       // sort color in this box along dimension dim:
276       Arrays.sort(imageColors, lower, upper+1, dim.comparator);
277       // find the median point:
278       int half = count / 2;
279       int nPixels, median;
280       for (median = lower, nPixels = 0; median < upper; median++) {
281         nPixels = nPixels + imageColors[median].cnt;
282         if (nPixels >= half)
283           break;
284       }
285       return median;
286     }
287
288     ColorNode getAverageColor() {
```

```
289       int rSum = 0;
290       int gSum = 0;
291       int bSum = 0;
292       int n = 0;
293       for (int i = lower; i <= upper; i++) {
294         ColorNode ci = imageColors[i];
295         int cnt = ci.cnt;
296         rSum = rSum + cnt * ci.red;
297         gSum = gSum + cnt * ci.grn;
298         bSum = bSum + cnt * ci.blu;
299         n = n + cnt;
300       }
301       double nd = n;
302       int avgRed = (int) (0.5 + rSum / nd);
303       int avgGrn = (int) (0.5 + gSum / nd);
304       int avgBlu = (int) (0.5 + bSum / nd);
305       return new ColorNode(avgRed, avgGrn, avgBlu, n);
306     }
307
308     public String toString() {
309       String s = this.getClass().getSimpleName();
310       s = s + " lower=" + lower + " upper=" + upper;
311       s = s + " count=" + count + " level=" + level;
312       s = s + " rmin=" + rmin + " rmax=" + rmax;
313       s = s + " gmin=" + gmin + " gmax=" + gmax;
314       s = s + " bmin=" + bmin + " bmax=" + bmax;
315       s = s + " bmin=" + bmin + " bmax=" + bmax;
316       return s;
317     }
318   }
319
320   // — color dimensions ————————————
321   // The main purpose of this enumeration class is to
322   // associate the color dimensions RED, GREEN, BLUE
323   // with the corresponding comparators.
324
325   enum ColorDimension {
326     RED (new Comparator<ColorNode>() {
327       public int compare(ColorNode colA, ColorNode colB) {
328         return colA.red - colB.red;
329       }}),
330     GREEN (new Comparator<ColorNode>() {
331       public int compare(ColorNode colA, ColorNode colB) {
332         return colA.grn - colB.grn;
333       }}),
334     BLUE (new Comparator<ColorNode>() {
335       public int compare(ColorNode colA, ColorNode colB) {
336         return colA.blu - colB.blu;
337       }});
338
339     public final Comparator<ColorNode> comparator;
340
341     ColorDimension(Comparator<ColorNode> cmp) {
```

```
342       this.comparator = cmp;
343     }
344   }
345
346   //——— utility methods ————
347
348   void listColorNodes(){
349     int i = 0;
350     for (ColorNode color : quantColors) {
351       IJ.write("  color " + i + ": " + color.toString());
352       i++;
353     }
354   }
355
356
357 } //class MedianCut
```

B.3.3 `ColorHistogram` (Class)

This utility class is used to compute the histogram of (unique) image colors.

```
 1 package color;
 2
 3 import ij.process.ColorProcessor;
 4 import java.util.Arrays;
 5
 6 public class ColorHistogram {
 7   int colorArray[] = null;
 8   int countArray[] = null;
 9
10   public ColorHistogram(ColorProcessor ip) {
11     this((int[]) ip.getPixels());
12   }
13
14   public ColorHistogram(int[] pixelsOrig) {
15     int N = pixelsOrig.length;
16     int[] pixelsCpy = new int[N];
17     for (int i = 0; i < N; i++) {
18       // remove possible alpha components
19       pixelsCpy[i] = 0xFFFFFF & pixelsOrig[i];
20     }
21     Arrays.sort(pixelsCpy);
22
23     // count unique colors:
24     int k = -1; // current color index
25     int curColor = -1;
26     for (int i = 0; i < pixelsCpy.length; i++) {
27       if (pixelsCpy[i] != curColor) {
28         k++;
29         curColor = pixelsCpy[i];
30       }
31     }
32     int nColors = k+1;
```

```
33
34      // tabulate and count unique colors:
35      colorArray = new int[nColors];
36      countArray = new int[nColors];
37      k = -1; // current color index
38      curColor = -1;
39      for (int i = 0; i < pixelsCpy.length; i++) {
40        if (pixelsCpy[i] != curColor) { // new color
41          k++;
42          curColor = pixelsCpy[i];
43          colorArray[k] = curColor;
44          countArray[k] = 1;
45        }
46        else {
47          countArray[k]++;
48        }
49      }
50    }
51
52    public int[] getColorArray() {
53      return colorArray;
54    }
55
56    public int[] getCountArray() {
57      return countArray;
58    }
59
60    public int getNumberOfColors() {
61      if (colorArray == null)
62        return 0;
63      else
64        return colorArray.length;
65    }
66
67    public int getColor(int index) {
68      return this.colorArray[index];
69    }
70
71    public int getCount(int index) {
72      return this.countArray[index];
73    }
74  }
```

B.3.4 `Median_Cut_Quantization` (Class)

This simple ImageJ plugin demonstrates the use of the `MedianCutQuantizer` class to quantize color images. The quantization process has two steps:

- First, a `ColorQuantizer` object is created from a given image using one of the constructor methods provided.
- Then this `ColorQuantizer` can be used to quantize the original image or

any other image using the same set of representative colors (color table).

```
1  import ij.ImagePlus;
2  import ij.plugin.filter.PlugInFilter;
3  import ij.process.ByteProcessor;
4  import ij.process.ColorProcessor;
5  import ij.process.ImageProcessor;
6  import color.ColorQuantizer;
7  import color.MedianCutQuantizer;
8
9  public class Median_Cut_Quantization implements PlugInFilter {
10   // specify the desired number of colors:
11   static int NCOLORS = 32;
12
13   public int setup(String arg, ImagePlus imp) {
14     return DOES_RGB + NO_CHANGES;
15   }
16
17   public void run(ImageProcessor ip) {
18     ColorProcessor cp = (ColorProcessor) ip.convertToRGB();
19
20     // create a quantizer object
21     ColorQuantizer quantizer = new MedianCutQuantizer(cp, NCOLORS);
22     int qColors = quantizer.countQuantizedColors();
23
24     // quantize to an indexed image
25     ByteProcessor idxIp = quantizer.quantizeImage(cp);
26     ImagePlus idxIm = new ImagePlus("Quantized Index Image (" +
           qColors + " colors)", idxIp);
27     idxIm.show();
28
29     // quantize to an RGB image
30     int[] rgbPixels = quantizer.quantizeImage((int[]) cp.getPixels())
           ;
31     ImageProcessor rgbIp =
32       new ColorProcessor(cp.getWidth(), cp.getHeight(), rgbPixels);
33     ImagePlus rgbIm =
34       new ImagePlus("Quantized RGB Image (" + qColors + " colors)" ,
             rgbIp);
35     rgbIm.show();
36
37   }
38 }
```

Bibliography

[1] Adobe Systems. "Adobe RGB (1998) Color Space Specification" (2005). http://www.adobe.com/digitalimag/pdfs/AdobeRGB1998.pdf.

[2] D. H. Ballard and C. M. Brown. "Computer Vision". Prentice Hall, Englewood Cliffs, NJ (1982).

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software* **22**(4), 469–483 (1996).

[4] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: two new techniques for image matching. In R. Reddy, editor, "Proceedings of the 5th International Joint Conference on Artificial Intelligence", pp. 659–663, Cambridge, MA (1977). William Kaufmann, Los Altos, CA.

[5] R. E. Blahut. "Fast Algorithms for Digital Signal Processing". Addison-Wesley, Reading, MA (1985).

[6] C. Boor. "A Practical Guide to Splines". Springer-Verlag, New York (2001).

[7] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing* **34**, 344–371 (1986).

[8] G. Borgefors. Hierarchical chamfer matching: a parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **10**(6), 849–865 (1988).

[9] J. E. Bresenham. A linear algorithm for incremental digital display of circular arcs. *Communications of the ACM* **20**(2), 100–106 (1977).

[10] E. O. Brigham. "The Fast Fourier Transform and Its Applications". Prentice Hall, Englewood Cliffs, NJ (1988).

[11] I. N. Bronshtein and K. A. Semendyayev. "Handbook of Mathematics". Springer-Verlag, Berlin, third ed. (2007).

[12] H. Bunke and P. S. P. Wang, editors. "Handbook of Character Recognition and Document Image Analysis". World Scientific, Singapore (2000).

[13] W. Burger and M. J. Burge. "ImageJ Short Reference for Java Developers" (2008). http://www.imagingbook.com.

[14] W. Burger and M. J. Burge. "Principles of Image Processing—Fundamental Techniques". Springer, New York (2009).

[15] K. R. Castleman. "Digital Image Processing". Prentice Hall, Upper Saddle River, NJ (1995).

[16] E. Catmull and R. Rom. A class of local interpolating splines. In R. E. Barnhill and R. F. Riesenfeld, editors, "Computer Aided Geometric Design", pp. 317–326. Academic Press, New York (1974).

[17] F. Chang and C. J. Chen. A component-labeling algorithm using contour tracing technique. In "Proceedings of the Seventh International Conference on Document Analysis and Recognition ICDAR2003", pp. 741–745, Edinburgh (2003). IEEE Computer Society, Los Alamitos, CA.

[18] F. Chang, C. J. Chen, and C. J. Lu. A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision, Graphics, and Image Processing: Image Understanding* **93**(2), 206–220 (February 2004).

[19] P. R. Cohen and E. A. Feigenbaum. "The Handbook of Artificial Intelligence". William Kaufmann, Los Altos, CA (1982).

[20] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. "Introduction to Algorithms". MIT Press, Cambridge, MA, second ed. (2001).

[21] R. O. Duda, P. E. Hart, and D. G. Stork. "Pattern Classification". Wiley, New York (2001).

[22] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. "Computer Graphics: Principles and Practice". Addison-Wesley, Reading, MA, second ed. (1996).

[23] W. Förstner and E. Gülch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. In A. Grün and H. Beyer, editors, "Proceedings, International Society for

Photogrammetry and Remote Sensing Intercommission Conference on the Fast Processing of Photogrammetric Data", pp. 281–305, Interlaken (June 1987).

[24] D. A. Forsyth and J. Ponce. "Computer Vision—A Modern Approach". Prentice Hall, Englewood Cliffs, NJ (2003).

[25] H. Freeman. Computer processing of line drawing images. *ACM Computing Surveys* **6**(1), 57–97 (March 1974).

[26] M. Gervautz and W. Purgathofer. A simple method for color quantization: octree quantization. In A. Glassner, editor, "Graphics Gems I", pp. 287–293. Academic Press, New York (1990).

[27] A. S. Glassner. "Principles of Digital Image Synthesis". Morgan Kaufmann Publishers, San Francisco (1995).

[28] R. C. Gonzalez and R. E. Woods. "Digital Image Processing". Addison-Wesley, Reading, MA (1992).

[29] P. Green. Colorimetry and colour differences. In P. Green and L. MacDonald, editors, "Colour Engineering", ch. 3, pp. 40–77. Wiley, New York (2002).

[30] E. L. Hall. "Computer Image Processing and Recognition". Academic Press, New York (1979).

[31] C. G. Harris and M. Stephens. A combined corner and edge detector. In C. J. Taylor, editor, "4th Alvey Vision Conference", pp. 147–151, Manchester (1988).

[32] P. S. Heckbert. Color image quantization for frame buffer display. *Computer Graphics* **16**(3), 297–307 (1982).

[33] P. S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California, Berkeley, Dept. of Electrical Engineering and Computer Science (1989). http://www.cs.cmu.edu/~ph/#papers.

[34] J. Holm, I. Tastl, L. Hanlon, and P. Hubel. Color processing for digital photography. In P. Green and L. MacDonald, editors, "Colour Engineering", ch. 9, pp. 179–220. Wiley, New York (2002).

[35] B. K. P. Horn. "Robot Vision". MIT-Press, Cambridge, MA (1982).

[36] P. V. C. Hough. Method and means for recognizing complex patterns. US Patent 3,069,654 (1962).

[37] M. K. Hu. Visual pattern recognition by moment invariants. *IEEE Transactions on Information Theory* **8**, 179–187 (1962).

[38] R. W. G. Hunt. "The Reproduction of Colour". Wiley, New York, sixth ed. (2004).

[39] J. Illingworth and J. Kittler. A survey of the Hough transform. *Computer Vision, Graphics and Image Processing* **44**, 87–116 (1988).

[40] International Color Consortium. "Specification ICC.1:2004-10 (Profile Version 4.2.0.0): Image Technology Colour Management—Architecture, Profile Format, and Data Structure" (2004). http://www.color.org/documents/ICC1v42_2006-05.pdf.

[41] International Electrotechnical Commission, IEC, Geneva. "IEC 61966-2-1: Multimedia Systems and Equipment—Colour Measurement and Management, Part 2-1: Colour Management—Default RGB Colour Space—sRGB" (1999). http://www.iec.ch.

[42] International Organization for Standardization, ISO, Geneva. "ISO 13655: 1996, Graphic Technology—Spectral Measurement and Colorimetric Computation for Graphic Arts Images" (1996).

[43] International Organization for Standardization, ISO, Geneva. "ISO 15076-1:2005, Image Technology Colour Management—Architecture, Profile Format, and Data Structure: Part 1" (2005). Based on ICC.1:2004-10.

[44] International Telecommunications Union, ITU, Geneva. "ITU-R Recommendation BT.709-3: Basic Parameter Values for the HDTV Standard for the Studio and for International Programme Exchange" (1998).

[45] B. Jähne. "Practical Handbook on Image Processing for Scientific Applications". CRC Press, Boca Raton, FL (1997).

[46] B. Jähne. "Digitale Bildverarbeitung". Springer-Verlag, Berlin, fifth ed. (2002).

[47] A. K. Jain. "Fundamentals of Digital Image Processing". Prentice Hall, Englewood Cliffs, NJ (1989).

[48] X. Y. Jiang and H. Bunke. Simple and fast computation of moments. *Pattern Recognition* **24**(8), 801–806 (1991).

[49] L. Kitchen and A. Rosenfeld. Gray-level corner detection. *Pattern Recognition Letters* **1**, 95–102 (1982).

[50] D. G. Lowe. Object recognition from local scale-invariant features. In "Proceedings of the 7th IEEE International Conference on Computer Vision ICCV'99", vol. 2, pp. 1150–1157, Kerkyra, Corfu, Greece (1999). IEEE Computer Society, Los Alamitos, CA.

[51] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In P. J. Hayes, editor, "Proceedings of the 7th International Joint Conference on Artificial Intelligence IJCAI'81", pp. 674–679, Vancouver, BC (1981). William Kaufmann, Los Altos, CA.

[52] S. Mallat. "A Wavelet Tour of Signal Processing". Academic Press, New York (1999).

[53] E. H. W. Meijering, W. J. Niessen, and M. A. Viergever. Quantitative evaluation of convolution-based methods for medical image interpolation. *Medical Image Analysis* **5**(2), 111–126 (2001). http://imagescience.bigr.nl/meijering/software/transformj/.

[54] D. P. Mitchell and A. N. Netravali. Reconstruction filters in computer-graphics. In R. J. Beach, editor, "Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH'88", pp. 221–228, Atlanta, GA (1988). ACM Press, New York.

[55] M. Nadler and E. P. Smith. "Pattern Recognition Engineering". Wiley, New York (1993).

[56] A. V. Oppenheim, R. W. Shafer, and J. R. Buck. "Discrete-Time Signal Processing". Prentice Hall, Englewood Cliffs, NJ, second ed. (1999).

[57] T. Pavlidis. "Algorithms for Graphics and Image Processing". Computer Science Press / Springer-Verlag, New York (1982).

[58] C. A. Poynton. "Digital Video and HDTV Algorithms and Interfaces". Morgan Kaufmann Publishers, San Francisco (2003).

[59] C. E. Reid and T. B. Passin. "Signal Processing in C". Wiley, New York (1992).

[60] D. Rich. Instruments and methods for colour measurement. In P. Green and L. MacDonald, editors, "Colour Engineering", ch. 2, pp. 19–48. Wiley, New York (2002).

[61] A. Rosenfeld and P. Pfaltz. Sequential operations in digital picture processing. *Journal of the ACM* **12**, 471–494 (1966).

[62] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(5), 530–535 (May 1997).

[63] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision* **37**(2), 151–172 (2000).

[64] M. Seul, L. O'Gorman, and M. J. Sammon. "Practical Algorithms for Image Analysis". Cambridge University Press, Cambridge (2000).

[65] L. G. Shapiro and G. C. Stockman. "Computer Vision". Prentice Hall, Englewood Cliffs, NJ (2001).

[66] G. Sharma and H. J. Trussell. Digital color imaging. *IEEE Transactions on Image Processing* **6**(7), 901–932 (1997).

[67] Y. Sirisathitkul, S. Auwatanamongkol, and B. Uyyanonvara. Color image quantization using distances between adjacent colors along the color axis with highest color variance. *Pattern Recognition Letters* **25**, 1025–1043 (2004).

[68] S. M. Smith and J. M. Brady. SUSAN—a new approach to low level image processing. *International Journal of Computer Vision* **23**(1), 45–78 (1997).

[69] M. Sonka, V. Hlavac, and R. Boyle. "Image Processing, Analysis and Machine Vision". PWS Publishing, Pacific Grove, CA, second ed. (1999).

[70] M. Stokes and M. Anderson. "A Standard Default Color Space for the Internet—sRGB". Hewlett-Packard, Microsoft, www.w3.org/Graphics/Color/sRGB.html (1996).

[71] S. Süsstrunk. Managing color in digital image libraries. In P. Green and L. MacDonald, editors, "Colour Engineering", ch. 17, pp. 385–419. Wiley, New York (2002).

[72] S. Theodoridis and K. Koutroumbas. "Pattern Recognition". Academic Press, New York (1999).

[73] E. Trucco and A. Verri. "Introductory Techniques for 3-D Computer Vision". Prentice Hall, Englewood Cliffs, NJ (1998).

[74] K. Turkowski. Filters for common resampling tasks. In A. Glassner, editor, "Graphics Gems I", pp. 147–165. Academic Press, New York (1990).

[75] T. Tuytelaars and L. J. Van Gool. Matching widely separated views based on affine invariant regions. *International Journal of Computer Vision* **59**(1), 61–85 (August 2004).

[76] D. Wallner. Color management and transformation through ICC profiles. In P. Green and L. MacDonald, editors, "Colour Engineering", ch. 11, pp. 247–261. Wiley, New York (2002).

[77] A. Watt. "3D Computer Graphics". Addison-Wesley, Reading, MA, third ed. (1999).

[78] A. Watt and F. Policarpo. "The Computer Image". Addison-Wesley, Reading, MA (1999).

[79] G. Wolberg. "Digital Image Warping". IEEE Computer Society Press, Los Alamitos, CA (1990).

[80] G. Wyszecki and W. S. Stiles. "Color Science: Concepts and Methods, Quantitative Data and Formulae". Wiley–Interscience, New York, second ed. (2000).

[81] S. Zokai and G. Wolberg. Image registration using log-polar mappings for recovery of large-scale similarity and projective transformations. *IEEE Transactions on Image Processing* **14**(10), 1422–1434 (October 2005).

Index

Symbols

⊛ (correlation operator) 259, 280
⊕ (dilation operator) 280
⊖ (erosion operator) 280
∗ (convolution operator) 75, 214, 280
¬ (logic operator) 24, 280
∧ (logic operator) 24, 280
⌊⌋ 280
∂ 70, 280
∇ 280
`&` (operator) 87
`&&` (operator) 10, 58, 81
`|` (operator) 87
`>>` (operator) 87

A

accumulator array 54
`add` (method) 285, 291, 298
`addNumericField` (method) 295
adjoint matrix 200
Adobe
– RGB 111, 112
affine mapping 195, 205
`AffineMapping` (class) 243, 244
`AffineTransform` (class) 238
aliasing 141, 142, 148, 151, 152, 164, 234
ambient lighting 102
amplitude 127, 128
angular frequency 126, 127, 147, 152, 158
anonymous class 308
`apply` (method) 121, 122
`applyTable` (method) 83, 299
`applyTo` (method) 240, 242, 246, 252, 253
approximation 222
Arctan function 40, 165, 205, 280
area
– polygon 34
– region 34
`ArrayList` (class) 80, 285, 290
`Arrays.sort` (method) 309
`atan2` (method) 280
AWT 118

B

bandwidth 143
Bartlett window 170, 173, 174
`BasicStroke` (class) 292, 293
basis function 147, 149–151, 158, 164, 183, 184, 190
bias problem 61
bicubic interpolation 229
`BicubicInterpolator` (class) 250, 252
bilinear
– interpolation 227
– mapping 203, 205
`BilinearInterpolator` (class) 249, 252
`BilinearMapping` (class) 246
binary
– image 5
bitmap image 27
`black` (constant) 292
`BLUE` (constant) 308
bounding box 35
Bradford model 113, 116
`BradfordAdaptation` (class) 121

breadth-first 9
Bresenham algorithm 64

C

C2-continuous 220
card 280, 281
cardinal spline 218, 221
cardinality 280, 281
Catmull-Rom interpolation 221
`centralMoment` (method) 39
centroid 36
chain code 28, 34
chamfer
– algorithm 271
– matching 274
chromatic adaptation 111
– Bradford model 113, 116
– XYZ scaling 112
`ChromaticAdaptation` (class) 121
CIE 98
– chromaticity diagram 99, 102
– L*a*b* 104, 105
– standard illuminant 101
– XYZ 98, 104, 105, 108, 119, 123
circle 64, 198
circularity 34
circumference 33
city block distance 270
`clone` (method) 241
`Cloneable` (interface) 240
clutter 275
`collectCorners` (method) 80
`Collections` (class) 81
collision 15
`Color` (class) 118, 119, 292, 293
color
– difference 105
– image 85–124
– management 123
– temperature 101
color quantization 85–95, 301
– 3:3:2 86
– median-cut 88, 301–311
– octree 89
– populosity 88
color space 123
– colorimetric 97–123
– HSB 119
– HSV 119
– in Java 114
– L*a*b* 104
– sRGB 106
– XYZ 98
`ColorBox` (class) 304, 306
`ColorDimension` (class) 308
`ColorModel` (class) 118
`ColorNode` (class) 304, 305
`ColorProcessor` (class) 309
`ColorQuantizer` (interface) 301, 311
`ColorSpace` (class) 117–119, 121
comb function 139
compactness 34
`compareTo` (method) 81, 295
comparing images 255–278
`Complex` (class) 155
complex number 128, 282
`computeMatch` (method) 266, 268
computer
– graphics 2
– vision 3
`concat` (method) 242, 246
conic section 198
connected components problem 16
container 79
contour 17–26
`ContourOverlay` (class) 25
`ContourTracer` (class) 22
`convertToFloat` (method) 297
`convertToRGB` (method) 311
convex hull 35, 47
convexity 35
convolution 177, 259
– property 137, 175
`convolve` (method) 78
`Convolver` (class) 78, 299
coordinate
– Cartesian 194
– homogeneous 194, 241
`copyBits` (method) 300
`Corner` (class) 79, 81
corner 69
– detection 69–84
– point 84
– response function 71, 73
– strength 72
`CorrCoeffMatcher` (class) 266, 267
correlation 177, 259
– coefficient 260
cosine function 134
– one-dimensional 126
– two-dimensional 160, 161
cosine transform 183
cosine^2 window 173, 174
cross correlation 259–261
`CS_CIEXYZ` (constant) 120
`CS_GRAY` (constant) 120

`CS_LINEAR_RGB` (constant) 120
`CS_PYCC` (constant) 120
`CS_sRGB` (constant) 119, 120
`CS_sRGBt` (constant) 121
cubic
– B-spline interpolation 222
– interpolation 217
– spline 219
cycle length 126

D

D50 101, 102, 119
D65 102, 104, 107
DCT 183–190
– one-dimensional 183, 186
– two-dimensional 187
`DCT` (method) 186
deconvolution 180
delta function 137
depth-first 7
derivative
– first 77
determinant 200
DFT 144–183, 280
– one-dimensional 144–154
– two-dimensional 157–183
`DFT` (method) 155
diameter 35
Dirac function 133, 137
discrete
– cosine transform 183–190
– Fourier transform 144–183, 280
– sine transform 183
distance 82, 258
– city block 270
– Euclidean 258, 270
– Manhattan 270
– mask 271
– maximum difference 258
– sum of differences 258
– sum of squared differences 258
– transform 270
`DOES_RGB` (constant) 311
dots per inch (dpi) 153
dpi 153
`draw` (method) 83, 293
`drawLine` (method) 83, 296
DST 183
`duplicate` (method) 83, 241, 284, 297

E

eccentricity 42, 48
edge
– map 49
– strength 71
eigenvalue 42, 71
eigenvector 71
ellipse 42, 66, 198
`Ellipse2D` (class) 286
elliptical window 172
elongatedness 42
`enum` type 308
Euclidean distance 82, 265, 270
Euler number 45
Euler's notation 128
EXIF 107

F

fast Fourier transform 155, 162, 175, 177
fax encoding 28
feature 32
FFT *see* fast Fourier transform
filter
– Gaussian 70, 77
– in frequency space 175
– inverse 178
– linear 75
`Find_Corners` (plugin) 84
`findCorners` (method) 83
`FloatProcessor` (class) 266
flood filling 6–10
floor function 281
four-point mapping 197
Fourier 130
– analysis 130
– coefficients 130
– descriptor 32
– integral 130
– series 130
– spectrum 32, 131, 144
– transform 126–280
– transform pair 132, 134, 135
frequency 127, 152
– angular 126, 127, 147, 158
– common 127
– directional 164
– effective 164
– fundamental 130, 152, 153
– maximum 142, 164
– space 132, 152, 175
– two-dimensional 164
`fromCIEXYZ` (method) 116–118, 121
function
– basis 147, 149–151, 158
– cosine 126
– delta 137

– Dirac 133, 137
– impulse 133, 137
– periodic 126
– sine 126
fundamental
– frequency 130, 152, 153
– period 152

G

gamma correction 114, 120, 122
– modified 108
gamut 102, 106, 111
– Adobe RGB 112
– sRGB 112
Gaussian
– area formula 34
– filter 70, 77
– function 133, 135
– window 170, 172, 174
`GenericDialog` (class) 295
geometric operation 191–254
`get` (method) 286, 289
`getComponents` (method) 119
`getf` (method) 267, 268
`getInterpolatedPixel` (method) 249–251
`getInverse` (method) 240
`getMagnification` (method) 293
`getMatchValue` (method) 268
`getNextNumber` (method) 295
`getPixel` (method) 87
`getPixels` (method) 297
`getTitle` (method) 294
GIF 28
gradient 70, 77
graph 16
`Graphics` (class) 292
graphics overlay 25
`Graphics2D` (class) 293
grayscale
– conversion 110
`GREEN` (constant) 308

H

Hadamard transform 188
Hanning window 170, 171, 173, 174
Harris corner detector 70
`HarrisCornerDetector` (class) 78, 84
`hasNext` (method) 285
Hertz 127, 152
Hessian normal form 54, 62
histogram 281
homogeneous
– coordinate 194, 241
Hough transform 50–67
– bias problem 61
– edge strength 63
– for circles 64–66
– for ellipses 66–67
– for lines 50–63
– generalized 67
– hierarchical 63
`HSBtoRGB` (method) 119
HSV 119

I

i (imaginary unit) 128, 281, 282
ICC 116
– profile 121
`ICC_ColorSpace` (class) 120, 123
`ICC_Profile` (class) 123
`iDCT` (method) 186
`Illuminant` (class) 121
illuminant 101
image
– binary 5
– coordinates 281
– space 175
– warping 204
`ImageCanvas` (class) 292
ImageJ
– geometric operation 238
`ImagePlus` (class) 84, 292, 311
`ImageWindow` (class) 284
impulse
– function 133, 137
in place 159
`IndexColorModel` (class) 304
`Integer.MAX_VALUE` (constant) 286
interest point 69
interpolation 210–233, 248–251
– B-spline 221, 222
– bicubic 229, 233, 250
– bilinear 227, 232, 249
– by convolution 217
– Catmull-Rom 219, 221, 251
– cubic 217
– ideal 213
– kernel 217
– Lanczos 223, 231, 254
– linear 217
– Mitchell-Netravali 221, 222, 254
– nearest-neighbor 217, 226, 232, 236, 249
– spline 219
– two-dimensional 225–233
`invalidNumber` (method) 295

invariance 34, 37, 38, 43, 45, 256
inverse
– filter 178
`invert` (method) 240, 242, 246
`isLocalMax` (method) 81
`isNaN` (method) 287
isotropic 70, 84
`Iterator` (class) 285
`iterator` (method) 285
ITU709 107

J

`Jama` (package) 199, 246, 247
JPEG 28, 95, 107, 109, 187

L

L*a*b* 104
`Lab_ColorSpace` (class) 117, 121, 122
label 6
Lanczos interpolation 223, 231, 254
line
– endpoints 62
– equation 51, 54
– Hessian normal form 54
– intercept/slope form 51
– intersection 62
linear
– convolution 75
– interpolation 217
– transformation 199
linearity 136
`LinearMapping` (class) 241, 244
`LinkedList` (class) 9, 291
`List` (interface) 80, 285, 288, 291, 292, 297
list 279
local mapping 207
lookup table 83, 299

M

major axis 38
`makeInverseMapping` (method) 248
`makeMapping` (method) 243, 245
Manhattan distance 270
`Mapping` (class) 239
mapping
– affine 195, 205
– bilinear 203, 205
– four-point 197
– function 193
– linear 199
– local 207
– nonlinear 204
– perspective 198
– projective 197–203, 205
– ripple 206
– spherical 207
– three-point 195
– twirl 204
mask 26
`Matrix` (class) 247
maximum
– frequency 142, 164
media-oriented color 109
median-cut algorithm 88, 301
`MedianCutQuantizer` (class) 301, 311
mesh partitioning 207
Mitchell-Netravali interpolation 222, 254
mod operator 154, 214, 281
moment 28, 37–44
– central 37
– Hu's 43, 48
– invariant 43
– least inertia 38
`moment` (method) 39
morphing 208
`MULTIPLY` (constant) 300

N

`NaN` (constant) 286
nearest-neighbor interpolation 217
`NearestNeighborInterpolator` (class) 249
neighborhood 6, 33
neutral
– point 101
`next` (method) 285
`NO_CHANGES` (constant) 311
nonmaximum suppression 59
`normalCentralMoment` (method) 39
Nyquist 143, 164

O

object 280
OCR 32, 46
octree algorithm 89
orientation 38, 164, 165
orthogonal 190
oscillation 126, 127
overlay 284

P

parameter space 51
Parzen window 170, 171, 173, 174
pattern recognition 3, 32

perimeter 33
period 126
periodicity 126, 158, 163, 167
perspective
– image 66
– mapping 198
phase 127, 153
– angle 128
`PixelInterpolator` (class) 249
Plessey detector 70
PNG 107
`Point` (class) 253, 285, 286, 289
`point` (class) 22
`Point2D` (class) 239, 241, 253
`Point2D.Double` (class) 287
`Polygon` (class) 286, 293
polygon
– area 34
`pop` (method) 10
populosity algorithm 88
power spectrum 153, 162
print pattern 181
profile connection space 115, 119
projection 44, 48
projective mapping 197–203, 205
`ProjectiveMapping` (class) 244, 252
pseudo-perspective mapping 198
`push` (method) 10

Q

quadrilateral 197
quantization 85–95
– linear 86
– scalar 86
– vector 88
`quantizeImage` (method) 311

R

`Rectangle` (class) 287
rectangular pulse 133, 135
– window 172
`RED` (constant) 308
refraction index 207
region 5–48
– area 34, 38, 48
– centroid 36, 48
– convex hull 35
– diameter 35
– eccentricity 42
– labeling 6–17
– major axis 38
– matrix representation 26
– moment 37
– orientation 38
– perimeter 33
– projection 44
– run length encoding 27
– topology 45
relative colorimetry 112
`RenderingHints` (class) 293
resampling 209–210
`RGBtoHSB` (method) 119
ripple mapping 206
`Rotation` (class) 244, 252
rotation 43, 177, 191, 193, 251
round function 281
roundness 34
run length encoding 27

S

sampling 137–143
– frequency 164
– interval 140, 141
– theorem 141, 143, 148, 151, 164, 213
`scale` (method) 293
`Scaling` (class) 244
scaling 43, 191, 193
separability 187
sequence 279
set 279
`setColor` (method) 293
`setf` (method) 268
`setMinAndMax` (method) 284
`setNormalize` (method) 78, 299
`setRenderingHint` (method) 293
`setStroke` (method) 293
`setup` (method) 284, 294
`setValue` (method) 83, 296
Shah function 139
Shannon 143
`Shape` (class) 286, 292, 293
shape
– feature 32
– number 30, 31, 47
`Shear` (class) 244
shearing 193
shift property 136
`show` (method) 84, 299, 311
`showDialog` (method) 295
`showMessage` (method) 295
`showProcessor` (method) 299
signal space 132, 152
similarity 136
Sinc function 133, 214, 225
sine function 134
– one-dimensional 126

sine transform 183
`size` (method) 285
`solve` (method) 247
`sort` (method) 81, 298, 309
source-to-target mapping 209
spectrum 125–190
spherical mapping 207
spline
– cardinal 218, 221
– Catmull-Rom 219, 221, 222
– cubic 219, 222
– cubic B- 221, 222, 253
– interpolation 219
`sqr` (method) 299
square window 174
sRGB 106, 108, 112, 114
– ambient lighting 102
– grayscale conversion 110
– white point 102
`Stack` (class) 9, 10
stack 7
standard illuminant 101, 111
`Stroke` (class) 293
structure 280
`super` (method) 292
super-Gaussian window 170, 172

T

target-to-source mapping 204, 210, 240
template matching 255, 257, 267
three-point mapping 195
threshold 59
TIFF 28
time unit 127
`toArray` (method) 81, 298
`toCIEXYZ` (method) 116–119, 122
topological property 45
tracking 69
transform pair 132
`TransformJ` (package) 238
`translate` (method) 286, 293
`Translation` (class) 244
translation 43, 193
tree 7
truncate function 281
tuple 280
twirl mapping 204
`TwirlMapping` (class) 247
`TYPE_Lab` (constant) 121

U

unit square 204

V

variance 261
`Vector` (class) 80, 298
vector 279
– graphics 25
viewing angle 102

W

Walsh transform 188
warping 204
`wasCanceled` (method) 295
wave number 147, 158, 164, 184
wavelet 188
`white` (constant) 292
white point 101, 104
– D50 101, 116
– D65 102, 107
windowed matching 267
windowing 169
windowing function 169–171
– Bartlett 170, 173, 174
– cosine^2 173, 174
– elliptical 170, 172
– Gaussian 170, 172, 174
– Hanning 170, 173, 174
– Parzen 170, 173, 174
– rectangular pulse 172
– super-Gaussian 170, 172

X

XYZ scaling 112

About the Authors

Wilhelm Burger received a Master's degree in Computer Science from the University of Utah (Salt Lake City) and a doctorate in Systems Science from Johannes Kepler University in Linz, Austria. As a post-graduate researcher at the Honeywell Systems & Research Center in Minneapolis and the University of California at Riverside, he worked mainly in the areas of visual motion analysis and autonomous navigation. In the Austrian research initiative on digital imaging, he was engaged in projects on generic object recognition and biometric identification. Since 1996, he has been the director of the Digital Media degree programs at the Upper Austria University of Applied Sciences at Hagenberg. Personally the author appreciates large-engine vehicles and (occasionally) a glass of dry "Veltliner".

Mark J. Burge received a BA degree from Ohio Wesleyan University, a MSc in Computer Science from the Ohio State University, and a doctorate from Johannes Kepler University in Linz, Austria. He spent several years as a researcher in Zürich, Switzerland at the Swiss Federal Institute of Technology (ETH), where he worked in computer vision and pattern recognition. As a post-graduate researcher at the Ohio State University, he was involved in the "Image Understanding and Interpretation Project" sponsored by the NASA Commercial Space Center. He earned tenure within the University System of Georgia as an associate professor in computer science and served as a Program Director at the National Science Foundation. Currently he is a Principal at Noblis (Mitretek) in Washington D.C. Personally, he is an expert on classic Italian espresso machines.

About this Book Series

The complete manuscript for this book was prepared by the authors "camera-ready" in LaTeX using Donald Knuth's Computer Modern fonts. The additional packages algorithmicx (by Szász János) for presenting algorithms, listings (by Carsten Heinz) for listing progam code, and psfrag (by Michael C. Grant and David Carlisle) for replacing text in graphics were particularly helpful in this task. Most illustrations were produced with Macromedia Freehand (now part of Adobe), function plots with Mathematica, and images with ImageJ or Adobe Photoshop. All book figures, test images in color and full resolution, as well as the Java source code for all examples are available at the book's support site: www.imagingbook.com.