Developing a raster detector system with the J array processing language

by Jan Jacobs

All digital copying aims to reproduce an original image as faithfully as possible under certain constraints. In the past, image processing had to be implemented in hardware for performance reasons. Here, a 100% software solution is outlined. In order to find such a solution an appropriate methodology based on the array processing language J is used. Although J is ideal for prototyping such designs, its wider application is seriously hindered by the lack of awareness of array processing languages amongst engineers, and by the lack of available education in this language and methodology.

Digital copying involves a process called half-toning, which transforms a continuous range of tones into a two-tone image. Text and image have different half-toning procedures, and a relative simple edge detector can differentiate between them. However, raster images confuse detectors by letting them think erroneously that text instead of image has to be half-toned. The consequence of this is shown by the 'lighter' blobs in the clipped halftone result (shown by a rectangle in Fig. 1).

This article describes the development of a local raster detector to solve this problem, using a proven methodology using the relatively new array processing language J.

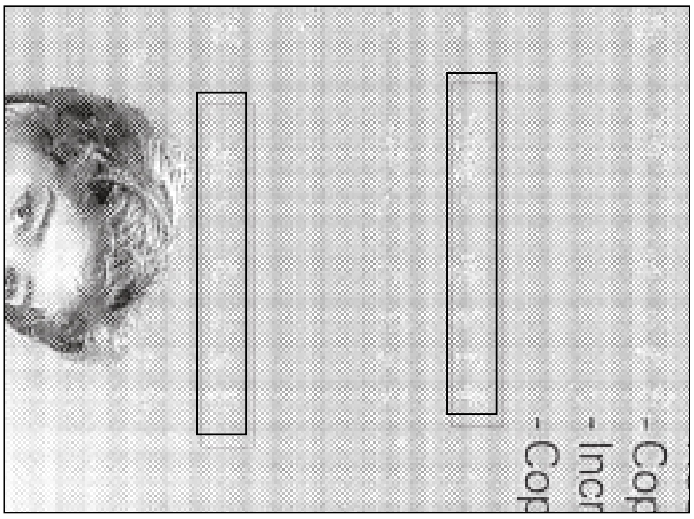


Fig. 1 Clipped part of a problematic half-toning of raster information

Towards system requirements

In principle it is possible to detect the existence of a raster in a local neighbourhood by, for example, using a Fourier transform. However, developing a detector on a standard PC platform requires that adequate quality is achieved with a computation time for detection, which does not exceed 0·5s for an A4 original scanned at 300 dpi (dots per inch, a measure for printing resolution) resolution. Also the detection should be independent of the raster form (e.g. line, dot rasters) and orientation. It should also be based on information retrieved locally, and it has been determined that 8×8 pixel sized tiles should be adequate. Since only grey level images are considered, pixels may be taken as equivalent to bytes.

The frequency band causing half-toning problems is 60-135 lpi (lines per inch, a measure for raster frequency), and another problem is caused by the erroneous detection of text as a low-frequency raster. It has been empirically

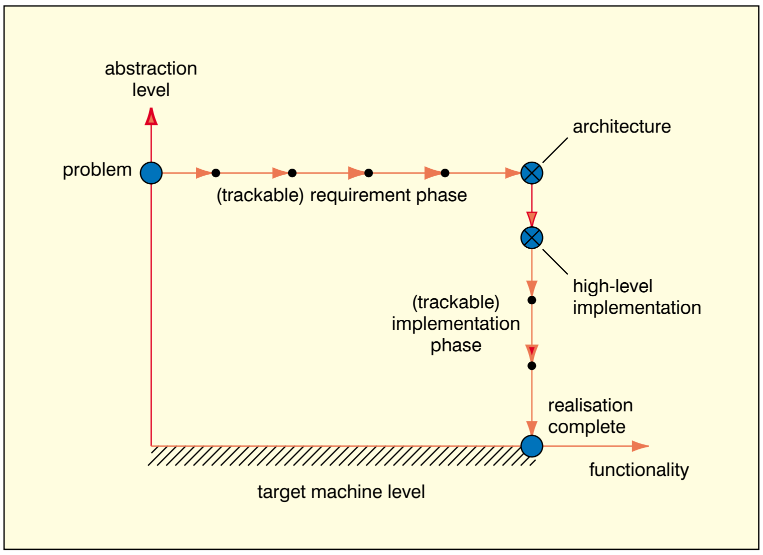


Fig. 2 Used development methodology; separation of functionality and implementation

determined that text builds up most power in the DC and 37 lpi frequency bands in 8 × 8 tiles on 300 dpi scans.

Finally, pure horizontal or vertical line pairs should not be treated as rasters. The human eye is particularly sensitive for these kind of patterns, and the best way is to treat the line pairs as text.

In summary, a solution has to be found which meets the following requirements:

- Computation time must be under 0·5s on a 600MHz Pentium III processor.
- Only 60-135 lpi rasters, independent of raster form and orientation, should be detected.
- Text should not be classified as a raster.
- Strict horizontal or vertical line rasters should not be classified as a raster.

Approach taken

The methodology supports a traditional incremental design process but with one difference, namely that intermediate designs are executable.

For the *requirements phase* this means that early feedback is possible and that functional specifications can be agreed on in an early stage (see Fig. 2, where the executable specs are denoted by 'architecture'). The actual architecture for *rasterDetect* will be derived later.

For the *implementation phase* early feedback on design alternatives and (cumulative) design choices minimise proliferation of errors. A first implementation is outlined later (see also Fig. 2).

In the development process, choices are recorded for both phases. This is done in such way that, at all major decision points, choices can easily be undone to support changes in design.

The approach is based on the 'computer architecture' methodology for hardware and software systems developed by Blaauw and Brooks while working at IBM in the late 1960s.[1] At that time the language APL was used to support the methodology.

Here the successor language J is used to support both the problem analysis and the design process. J fragments will be given that helped in developing the architecture as well as the implementation process in an interactive way. The code given in these fragments is accompanied by examples. The reader should note that, because of the Journal column widths, the J fragments have been 'wrapped around' and would normally appear on one line.

149	163	149	154	152	143	163	150	164	146	159	143	161	145	164	148
158	147	159	156	154	156	150	157	145	163	147	164	146	158	152	157
147	165	148	154	152	146	159	147	159	147	157	150	158	147	163	150
157	145	156	154	161	152	150	153	148	157	148	161	149	154	152	153
148	166	147	156	152	152	157	150	156	152	155	155	157	150	156	154
158	152	156	154	156	151	155	150	153	159	151	150	155	149	161	146
150	159	149	156	154	152	154	153	149	157	150	156	150	157	149	157
157	156	154	153	154	151	153	152	155	152	152	150	159	146	164	146
150	153	151	151	155	151	155	155	148	158	150	158	148	156	142	160
151	156	149	153	150	153	153	154	150	150	152	145	161	142	168	140
151	153	156	149	158	149	156	152	151	157	148	161	147	159	140	164
148	157	147	153	150	155	152	155	151	150	153	146	162	143	169	142
152	151	157	148	159	148	153	151	149	158	149	164	146	160	140	166
145	160	144	154	146	158	151	159	152	150	154	141	166	143	173	139
158	142	163	144	165	142	154	148	149	159	150	163	145	162	134	168
143	165	137	160	144	160	150	158	151	148	154	141	165	141	173	135

Fig. 3 Bitmap read into the matrix variable

Architecture

In this phase the required functionality of the detector is developed. It is coded in an executable specification (architecture) without the burden of implementation constraints. The analysis as well as the construction of the architecture is supported by J code fragments.

Problem analysis

Promising methods found in the literature to detect rasters are often based on Fourier transforms to solve the following problems:

- independence of raster form (dots, line pairs etc.)
- independence of raster orientation (all angles)
- handling of text

- handling of strict horizontal or vertical line rasters.

Execution speed is not an issue in this phase.

To start the investigation, a part of the problem area in Fig.1 is read in a matrix.

```
load ‘graph’ NB. load image
library
bitmap =: readbmp ‘raster.bmp’
NB. read bitmap
```

A small bitmap (16×16) is read into matrix variable bitmap (Fig. 3).

Since the analysis must be performed in a local environment the bitmap is first tiled in 8×8 byte patches before being transformed into the frequency domain.

149	163	149	154	152	143	163	150	164	146	159	143	161	145	164	148
158	147	159	156	154	156	150	157	145	163	147	164	146	158	152	157
147	165	148	154	152	146	159	147	159	147	157	150	158	147	163	150
157	145	156	154	161	152	150	153	148	157	148	161	149	154	152	153
148	166	147	156	152	152	157	150	156	152	155	155	157	150	156	154
158	152	156	154	156	151	155	150	153	159	151	150	155	149	161	146
150	159	149	156	154	152	154	153	149	157	150	156	150	157	149	157
157	156	154	153	154	151	153	152	155	152	152	150	159	146	164	146
150	153	151	151	155	151	155	155	148	158	150	158	148	156	142	160
151	156	149	153	150	153	153	154	150	150	152	145	161	142	168	140
151	153	156	149	158	149	156	152	151	157	148	161	147	159	140	164
148	157	147	153	150	155	152	155	151	150	153	146	162	143	169	142
152	151	157	148	159	148	153	151	149	158	149	164	146	160	140	166
145	160	144	154	146	158	151	159	152	150	154	141	166	143	173	139
158	142	163	144	165	142	154	148	149	159	150	163	145	162	134	168
143	165	137	160	144	160	150	158	151	148	154	141	165	141	173	135

Fig. 4 Result of tesselation

```
load ‘addons\fftw\fftw’
NB. load fft library
tiles =: (2 2$8) <;._3 bitmap
NB. tessellate bitmap
tilesF =: fftw &.> tiles
NB. apply fft analysis to each tile
```

The verb <;._3 is a shorthand in J for ‘tessellate’ with the tiling pattern given by the left argument. This tiling pattern, defined by 2 2$8, evaluates into a 2×2 shaped matrix containing all 8s, is given below:

```
8 8
8 8
```

This matrix actually defines a movement vector and a displacement vector for the two dimensions used in the tiling process. The result of this tessellation is given in Fig. 4.

The spectrum of one of these patches or tiles (upper left) is shown in Fig. 5. The following observations can be made:

- The (amplitude) spectrum contains complex numbers, the real part is at the *j*’s left-hand side, the imaginary part at the right-hand side.
- The DC component, upper left corner at (0,0), is relatively large.
- The maximum amplitude can be found at (*y*,*x*) = (4,3) and at (4,5) indicating a relatively high frequency raster.
- Note the symmetry around column and row number 4, the principal components may be confined into an upper left 5×5 sub-matrix.

Analysis indicates that in the problem regions two dominant rasters occur in the 60-135 lpi band. The solution of the problem is a band detector, which should have the circular symmetry property, since rasters may be positioned under arbitrary angle (see Fig. 6).

Towards a solution

The architecture is a functionally complete and

y \ x								
	9.82e3	6.68j_30	0j_7	_28.7j_76	13	_28.7j76	0j7	6.68j30
	_0.0503j6.88	15.5j7.03	_5.81j11.2	3.59j_1.51	3.91j_7.71	_7.17j4.9	_19.5j_4.54	17.5j6.83
	6j_11	8.19j0.293	_5j8	2.88j_25.6	2j3	_10.2j1.71	_11j12	7.12j_4.39
	_9.95j_11.1	6.41j18.5	21.5j_2.54	32.5j_22.1	41.1j6.29	_47.5j_1.17	_24.2j_26.8	_12.8j14.9
	_31	17.9j_3.1	_44j_45	_59.9j_117	_85	_59.9j117	_44j45	17.9j3.1
	_9.95j11.1	_12.8j_14.9	_24.2j26.8	_47.5j1.17	42.1j_6.29	32.5j22.1	21.5j2.54	6.41j_18.5
	6j11	7.12j4.39	_11j_12	_10.2j_1.71	2j_3	2.88j25.6	_5j_8	8.19j_0.293
	_0.0503j_6.88	17.5j_6.83	_19.5j4.54	_7.17j_4.9	3.9j7.71	3.59j1.51	_5.81j_11.2	15.5j_7.93

Fig. 5 Spectrum of one of the patches or tiles

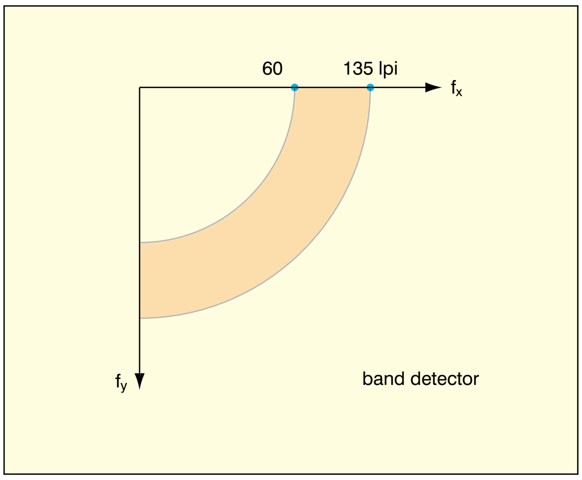
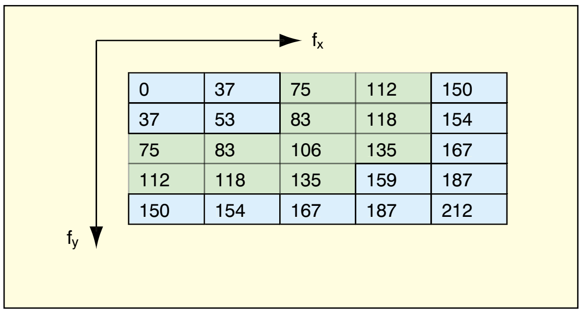


Fig. 6 Requested band detector



0	37	75	112	150
37	53	83	118	154
75	83	106	135	167
112	118	135	159	187
150	154	167	187	212

Fig. 7 Band detector approximation (numbers indicate lpi)

executable specification, and therefore provides a proven correct basis when starting the implementation process.

In an 8 × 8 tile, rasters as fine as 212 lpi can be detected, large enough for the required 60-133 lpi band (see Fig. 7). Since the 8 × 8 spectrum of this tile exhibits redundant information only the principal components—expressed by their lpi numbers—are presented in a 5 × 5 matrix. The band detector indicated by Fig. 6 is approximated by the shaded components.

The specification for raster detection in the 60-135 lpi interval, resulting in a single boolean per tile, is now derived and expressed in J fragments.

The matrix *singleTileF* is one of the many frequency domain tiles (*tilesF*, see Fig. 5). The power spectrum of this 8 × 8 tile is taken by first squaring each complex number (*:) followed by taking its magnitude (|). Note the right to left evaluation order, see Fig. 8. The identity function] forces a display of its right argument (assignments are silent).

```
] powerTileF =: |*: singleTileF
```

Since only the shaded components in Fig. 7 are required we only select the lower 4 × 4 submatrix (4 4{.), see Fig. 9:

```
] intermdF =: 4 4{. powerTileF
```

Looking closer to the components which have to be selected one can cover the shaded area by masking of the lower 2 × 2 sub-matrix and component (3,3). This is done by function ammend (}), which can update specified (group) components. The left 0 is the update value (see Fig. 10 for the filtered result):

```
] filteredF =: 0((0 1;0 1);3 3)} intermdF
```

The accumulated power in this frequency band is determined by counting all columns by function +/ into four sub-totals and again +/ to accumulate all column totals. Comparing it with a certain threshold returns the answer whether a raster is present or not:

```
rasterDetected=: THRESHOLD < +/+/
filteredF
```

Putting all the parts together will reveal the complete functional specification (the architecture) of this raster detector:

```
RasterDetected =: THRESHOLD < +/+/
0((0 1;0 1);3 3)} 4 4{. |*: singleTileF
```

The rejection of pure horizontal and vertical rasters is not given here but is based on the typical power distribution between DC and accumulated AC bands.

Implementation

Although the specified functionality is executable it is still far a way from the target language (in our case C). The goal of the implementation phase (see Fig. 2) is to work towards a realisation without compromising or altering the functionality (architecture). In this section only a few implementation problems are addressed.

Problem analysis

Implementation of raster detection in software on a standard PC with a processing time of less than 0·5s requires new techniques, such as:

9.65e7	947	49	6.6e3	169	6.6e3	49	947
47.3	304	160	15.2	74.7	75.4	401	354
157	67.2	89	664	13	107	265	70
223	383	469	1.54e3	1.81e3	2.26e3	1.3e3	387
961	330	3.96e3	1.73e4	7.23e3	1.73e4	3.96e3	330
223	387	1.3e3	2.26e3	1.81e3	1.54e3	469	383
157	70	265	107	13	664	89	67.2
47.3	354	401	75.4	74.7	15.2	160	304

Fig. 8 Power spectrum

- reduction of dimensionality
- reduction of data size
- use of look-up tables (LUTs) to speed up Fourier transforms
- reduction of spectrum information to only the two most dominant power bands.

The last two are not elaborated here. We will elaborate on the other two techniques:

- *Reduction of dimension:* Current PC processors are not equipped to deal with Fourier transforms of a single 2D tile. To do so the 2D problem must be transformed in real time to simpler multiple 1D problems.

- *Reduction of data size:* A good estimate of frequency content can be made by finding the number of times the grey values within each byte cross the average values of rows and columns (see Fig. 11).

Reduction of data size has to be performed for each tile, and within a tile for each of its rows and each of its columns. The following derivation shows the reduction for rows for a single tile (most upper left tile in the tiled bitmap *tiles*). This tile, again the upper left one, is given as Fig. 12.

The averages per row are computed via a so-called fork construct, a function composition. Averaging rows includes summation over rows (+/"1), counting the number of members (#) and taking the quotient of both (%); concisely notated by ((+/"1 % #)).

The average of each row is given below by vector *threshRows*:

```
] threshRows =: (+/"1 % #) singleTile
```

```
152.875 154.625 152.25 153.5 153.5 154 153.375
153.75
```

Comparing each row with this row average (*singleTile* > *threshRows*) will yield the result as described in Fig. 11 (see boolean matrix, Fig. 13):

```
] fingerPrintRows =: #. singleTile >
threshRows
```

The boolean row vectors within this matrix represent the essence of the original grey level information. To retrieve its confined raster information in a fast way, an index (to be used for a LUT) has to be produced for each of the rows. Each row is interpreted as a binary number by applying the function #., resulting in *fingerPrintRows*:

```
82 181 82 184 82 170 90 232
```

Summary

The problem size is reduced in several steps until it can be realised on a standard PC processor. Such reduction with acceptable quality degradation is performed in the following steps:

1 2D to multiple 1D transforms reduction
2 removal of DC power and other non relevant frequency content
3 reduction of grey level information to numbers of zero crossings
4 reduction of generated spectra to only the two highest power bands.

The complexity of this particular detector was sufficiently low that a C implementation could be made by hand with the necessary LUT tables generated by a J script to C source, although J could well have been used for the complete development: implementation as well as design.

9.65e7	947	49	6.6e3
47.3	304	160	15.2
157	67.2	89	664
223	383	469	1.54e3

Fig. 9 Intermediate result

0	0	49	6.6e3
0	0	160	15.2
157	67.2	89	664
223	383	469	0

Fig. 10 Filtered result

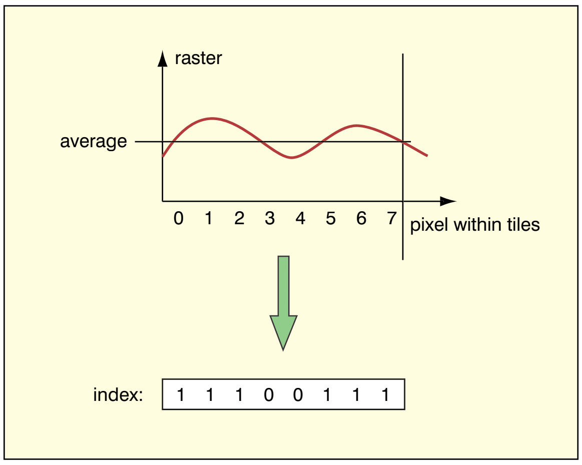


Fig. 11 Reducing the data size to 1 bit/pixel

149	163	149	154	152	143	163	159
158	147	159	156	154	156	150	157
147	165	148	154	152	146	159	147
157	145	156	154	161	152	150	153
148	166	147	156	152	152	157	150
158	152	156	154	156	151	155	150
150	159	149	156	154	152	154	153
157	156	154	153	154	151	153	152

Fig.12 Tile for 1 bit/pixel

0	1	0	1	0	0	1	0
1	0	1	1	0	1	0	1
0	1	0	1	0	0	1	0
1	0	1	1	1	0	0	0
0	1	0	1	0	0	1	0
1	0	1	0	1	0	1	0
0	1	0	1	1	0	1	0
1	1	1	0	1	0	0	0

Fig. 13 Boolean matrix

Conclusions

Raster detection

The goal of both raster form and orientation-independence is realised (see Fig. 14 to get an impression of the enhancements.) The white blobs are clearly much smaller than before.

The global raster detection is realised within 0·13 s on average per A4 page on a 600 MHz Pentium III. No significant improvement can be expected from optimisation in processing speed since within-memory data transfer rates enforce a lower bound, which has been determined as 0·09 s.

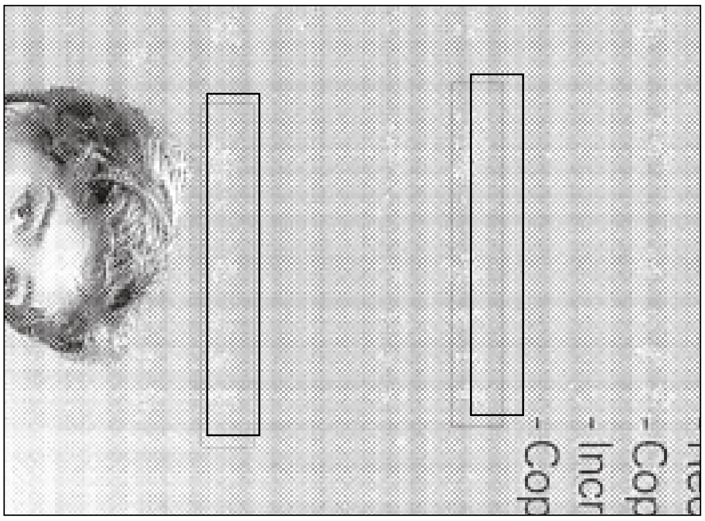


Fig. 14 Clipped part of the solved half-toning problem for raster information

Methodology and language issues

Using J for this and other applications helped build up experience of how IT professionals use the language. The following statements summarise their findings:

Positive features of J

1 It supports the requirement phase, starting from problem analysis to functional specification (architecture).
2 It supports the design and the recording of design decisions and alternatives during the implementation phase.
3 It enables early executability of specifications, and thereby reduces proliferation of errors further on in the software development process.
4 It enables functionality to be expressed at any given level of abstraction, from a high abstract level down to production code.

Negative features of J

1 The relationships between J concepts and the development methods used in the architectural and design processes are not completely straightforward.
2 More specifically, it is not immediately clear how to use the benefit of executable specifications, that is, by gradually replacing abstract specifications with implementation code.
3 Deriving an architecture and developing an implementation do not in themselves require a specific computing language.
4 J does not resemble other popular languages and has an esoteric image, which makes it difficult to switch between J and other languages.

To conclude, using J for problem analysis and system design is complicated by factors which have nothing to do with the language itself, but which can divert attention from the primary goal of putting a right methodology in place. Education is the way to address these issues, through courses at technical colleges and universities. However, for engineers who are willing to spend time applying this sort of methodology and learning the language J, the effort involved will certainly pay off.

Reference

1 BLAAUW, G. A., and BROOKS, JUN., F. P.: 'Computer architecture: concepts and evolution' (Addison-Wesley, Reading, MA, USA, 1997)

The author is with Océ Technologies BV, PO Box 101, 5900MA Venlo, Netherlands.