

DSC 2003 Working Papers
(Draft Versions)
http://www.ci.tuwien.ac.at/Conferences/DSC-2003/

R-Based Environment for Image Processing Algorithm Design

Tomomi TAKASHINA[*] **and Hanno ACKERMANN**[†]

Abstract

This manuscript reports the application of S for image processing, as one of the fields in which statistics isn't used explicitly. The conciseness and expressiveness of the S language are expected to support developers in image processing field strongly. The authors have developed rimage, an image processing library for R, and IBASE-0, an integrated database for image processing algorithm design. Library rimage is a collection of image processing functions in which C routines are used for speed improvement. In IBASE-0, an user can describe image operation in the S language and can apply such operations to the selected images through WWW browser. The authors conclude that R can be successfully integrated in practical development process in the image engineering field.

1 Introduction

Chambers mentioned that the overall goal for S is *to turn ideas into software, quickly and faithfully* [1]. This statement can hold true in the general engineering fields though S has been supported mainly by the fields where statistics is used explicitly such as biology, quality control, sociology, econometrics, and so on. S users in the general engineering fields have two merits. At first, users can make very compact representations which have good correspondence with their idea, that is one of the strong features of functional programming language. At second, users can utilize rich functions which deal with the real world such as statistics and numerical computation. So to speak, S is Lisp for ordinary engineers.

[*]Nikon Digital Technologies Co., Ltd., Japan
[†]University of Mannheim, Germany

For example, one of the authors developed an analysis tool in the field of Artificial Intelligence[6, 5]. The tool is implemented as an R library, which is for the information theoretical analysis of the behavior of intelligent agent. Users can concentrate on the analysis of their targets by the benefits of the both of the flexibility of R and the classes and functions which are provided by the library.

In the industrial field, 'algorithm' takes important role in much varieties of software. Software development projects often lack the viewpoint of independent algorithm development. Such projects might develop algorithm in the same language as that used in the production. However, such production level languages don't offer quick and flexible facilities for algorithm development.

For such a purpose, S is a strong environment. It is useful to implement algorithm quickly by S as Chambers mentioned. From the viewpoint, we adopt R as the key component of the environment for image processing algorithm design.

2 Image Processing Library for R

2.1 Image Processing by S

The advantages of image processing by S are the following.

1. Image processing is coded using abundant generic vector oriented functions that S has, because an image is represented in a matrix or an array. This approach makes the description of algorithm short and has more flexibility than the use of any image-specific library.

2. Developers can easily introduce many statistical techniques to their algorithms and evaluate those algorithms from the statistic point of view.

3. Interactive and automatizable operations makes experiments efficient. Developers can easily try any operations out and build routines using them, that is effective use of conciseness and expressiveness in functional programming style.

On the other hand, the disadvantage of image processing by S is that computation speed is slow. Generally, vector oriented functions in S are pretty fast in spite of the handicap that interpreters has generally. However, there are some operations that S is not good at, such as two dimensional convolution.

In the rest of this subsection, we show a few functions to see how concise and expressive S is in image processing. In the following examples, an image is represented in a matrix whose elements range from 0 to 1.

Listing 1 is thresholding. Modifying a subset of a matrix by a logical matrix is well suited to the situation.

Listing 1: Thresholding

```
thresholding <- function(img, th=0.5) {
  img[img < th] <- 0
  img[th <= img] <- 1
```

```
  img
}
```

Listing 2 is histogram equalization. Though vector style programming is different from typical sequential approach, it is elegant.

Listing 2: Histogram equalization

```
equalization <- function(img) {
  img.256 <- floor(255*img)
  ih.target <- 1/256
  ih.cum <- cumsum(hist(img.256, breaks=0:256, plot=F)$density)
  map <- floor(ih.cum/ih.target)
  apply(img.256, c(1,2), function(x) map[x+1])/255
}
```

Listing 3 is automatic thresholding using discriminant analysis. It's easy to read because the representation of `sum` and index vectors has good correspondence with $\sum$ in the formula of the algorithm. This code is not optimal. However, readability can have precedence in the case that the difference in speed is regarded to be trivial.

Listing 3: Automatic Thresholding

```
auto.thresholding <- function(img) {
  th.by.discrim <- function(img) {
    img.256 <- floor(255*img)
    h <- hist(img.256, breaks=0:256, plot=F)$density
    L <- 255
    lv <- 0:L
    u.img <- sum(lv * h) / sum(h)
    s.img <- sum(h * (lv - u.img)^2)
    Fs <- sapply(1:(L-1), function(k) {
      w.0 <- sum(h[1:k])
      w.1 <- sum(h[(k+1):L])
      u.0 <- sum((1:k) * h[1:k]) / w.0
      u.1 <- sum(((k+1):L) * h[(k+1):L]) / w.1
      s.B <- w.0 * (u.0 - u.img)^2 + w.1 * (u.1 - u.img)^2
      s.B / s.img
    })
    lv[rev(order(Fs, na.last = NA))[1]]/255
  }
  thresholding(img, th=th.by.discrim(img)))
}
```

We've conducted a comparison experiment for the functions, thresholding, histogram equalization, and automatic thresholding between S and C++. Please note that those functions don't include calculations where pure S is definitely weak, such as convolution.

We use those functions in Listing 1 $\sim$ 3 for S programs and the typical implementations for C++.

The result is shown in Table 1. It is clear that S is much expressive in image processing than C++ by comparing the length. The difference of length in thresholding

Table 1: Comparison of image processing between S and C++

	time* [sec]		length [lines]	
	S	C++	S	C++
thresholding	0.15	0.01	5	7
equalization	10.2	0.05	7	54(148)**
automatic thresholding	0.60	0.08	20	396(490)**

* Measurement was done on Linux 2.4.3 / Pentium III 933MHz / 512MB using R 1.6.1 and gcc-2.95.3 for S and C/C++ respectively. The image used for measurement is 8 bit depth and the size of 420x418.

** A number in parentheses includes the definition of a basic matrix class.

seems to be a little but Listing 1 shows S is much expressive. As for computation time, the thresholding and the automatic thresholding in S have enough speed. But the equalization in S needs so much time that is not practical. Though they are not enough sample, it shows that S image processing program is $8 \sim 200$ times slower.

It means that the speed improvement by C code is necessary even for algorithms that S isn't definitely weak in.

2.2 Development of rimage

We developed rimage[1] package which is a collection of image manipulating functions. In this package, an image is represented as a instance of imagematrix class which is a derivative of matrix and defined in rimage package.

In the earlier versions of rimage, we have used pixmap class which is defined in package pixmap 0.2 in order to represent image. Generic-function-based class is used for image representation in pixmap 0.2. The pixmap 0.3 changed image representation method to formal class. Finally, we invented imagematrix class because we wanted to use matrix functions as is for image instead of defining methods for formal class. For example, it is advantageous that we can write the following S sentence for applying gamma transformation to an image and displaying it.

```
plot(image^gamma)
```

C routines are used for speed improvement in the library. The functions the library provides are summarized in the Table 2.

[1] rimage 0.5 is available under BSD style license in CRAN.

Table 2: functions in rimage

Name	Description	Note
imagematrix	Generate an imagematrix object	
imageType	Get information on color type of imagematrix	
rgb2grey	Convert color imagematrix to grey imagematrix	
read.jpeg	Read JPEG file	Calling libjpeg[3]
plot.imagematrix	Plotting an imagematrix object	
print.imagematrix	Print information on a given imagematrix object	
clipping	Clipping image	
normalize	Normalization for vector and matrix	
equalize	Make image having equalized histogram	
fftImg	Compute FFT image	
fftw	Apply FFT to 2-Dimensional Data	Calling FFTW[2]
lowpass	Low Pass Filter for Image	C routine
highpass	High pass filter for image	C routine
meanImg	Mean filter	C routine
minImg	3x3 min filter	C routine
maxImg	3x3 max filter	C routine
laplacian	Laplacian of image	C routine
sobel	Sobel filter	C routine
thresholding	Thresholding image	

3 IBASE-0: Integrated Database Using R as Image Processing Engine

We developed integrated database which provides the verification facilities of the image processing algorithm using R as image processing engine.

The features of IBASE-0 are in the following.

1. Users can choose images by specifying search condition in WWW browser.
2. Users can apply various image operators to a set of images which are chosen by step 1 through WWW browser.
3. Users can easily enhance a set of image operators by the S language.

3.1 The Architecture of IBASE-0

The architecture of IBASE-0 is the combination of PostgreSQL as RDBMS, Apache + PHP as WWW application server, and R as data analysis environment as shown in Figure 1.

The communication between those components is shown in Figure 2. As for communication between PostgreSQL and R, we employed RPgSQL for normal SQL and developed dbimage for image transfer. The former was developed by T. H. Keitt and available in http://rpgsql.sourceforge.net/. The latter was developed by us for reading and writing large objects of PostgreSQL in the R environment. R is forked for every set of images to be processed in the server side. Though it seems to be waste of time, we decided it for making the system robust.

The database stores the following type of information.

images images in PGM or PNG format

attributes attributes used mainly for search such as category and group

features feature information of images in which users can add feature types freely

script image operators which are called from the WWW server and written in the S language (same format as Listing 1 $\sim$ 3)

3.2 The Scenarios

We assume the following scenarios as the typical usages of IBASE-0.

3.2.1 Scenario 1: Try-and-Error

A team member conducts experiments to test how a combination of basic image filters works on a set of images through WWW client.

Figure 3 shows an example. Those two images in the right column are selected by a condition in the front window. Then, normalization, equalization, min filter, and max filter were applied sequentially.

3.2.2 Scenario 2: Evaluation

A new interesting algorithm is discovered by genetic programming. The development team needs to check its effectiveness in broader examples. IBASE-0 supports this collaborative work. The algorithm can be registered to IBASE-0 and be quickly confirmed if works for another images.

Let's assume that the genetic programming engine generates S code in Listing 4. In the code, object `ft` is a tree representation of program by genetic programming. Then, object `ft` is executed by function `apply.ftree()`, which is elegantly written using function `get()` and recursion.

Listing 4: Genetic programming example

```
gp.result <-
function(img) {
  ft <-
    list("adf.max",
         list("adf.threshold4",
              list("adf.darkpixel",
                   list("adf.darkpixel",
                        list("adf.max",
                             list("adf.threshold4",
                                  list("adf.darkpixel",
                                       list("adf.diff",
                                            "bg",
                                            "img"),
                                       NULL),
                                  NULL),
                             NULL),
                        NULL),
                   NULL),
              NULL),
         NULL)
  apply.ftree(ft)
}
```

3.2.3 Scenario 3: Precise Analysis

A team member develops a new algorithm on R in the client side. Then the member registers the new algorithm to the database. Other team members check how it works for images they are interested in through the R library for IBASE-0 client and WWW browser.

If they want to check new algorithm precisely, they can use R library for IBASE-0 client. The following example gets "removeIllumination" from the database and applies it to image `cat.image` by function `apply.ibscript()`. Function `apply.ibscript()` is elegantly implemented using `parse()` and `eval()` in the client side library of IBASE-0. Then it plots the result. He or she can continue the investigation further.

```
x <- apply.ibscript("removeIllumination", cat.image)
```

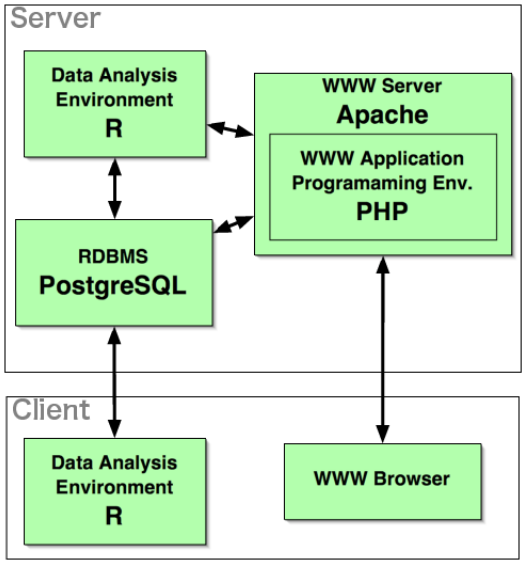


Figure 1: The architecture of IBASE-0

```
plot(x)
sum((x - cat.image)^2)
```

3.3 Response Time and Memory Consumption

We measured response time and memory consumption for a small scale problem.

We applied several operations for 10 images of the size of 480×640 pixels. The result is shown in Table 3.

It shows that IBASE-0 can be applied in a practical sense. Also, sensory response for users is better because those processed images are obtained one after another, which means that users don't have to wait result until all images are processed.

4 Discussion

As for rimage library, S is so expressive that one can write function very easily, which means that rimage doesn't need so many image processing functions. So the main purpose should be the performance improvement for slow computation in S. Further, we will add several C/C++ based functions to rimage in order to overcome slow computation.

As for IBASE-0, it's convenient that image operators written in S can be used from WWW without changes. As for future work, image search functions should be improved. Security problem should be also considered.

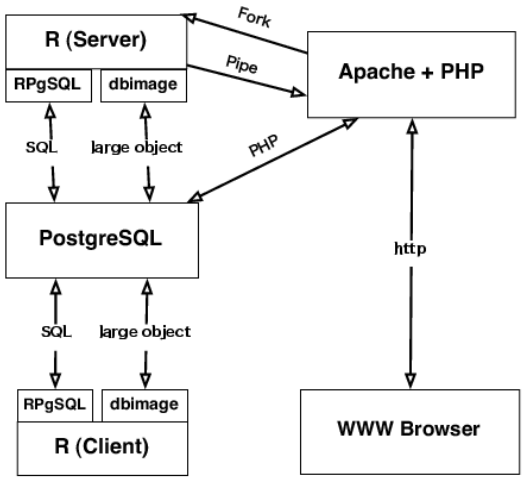


Figure 2: Communication between components in IBASE-0

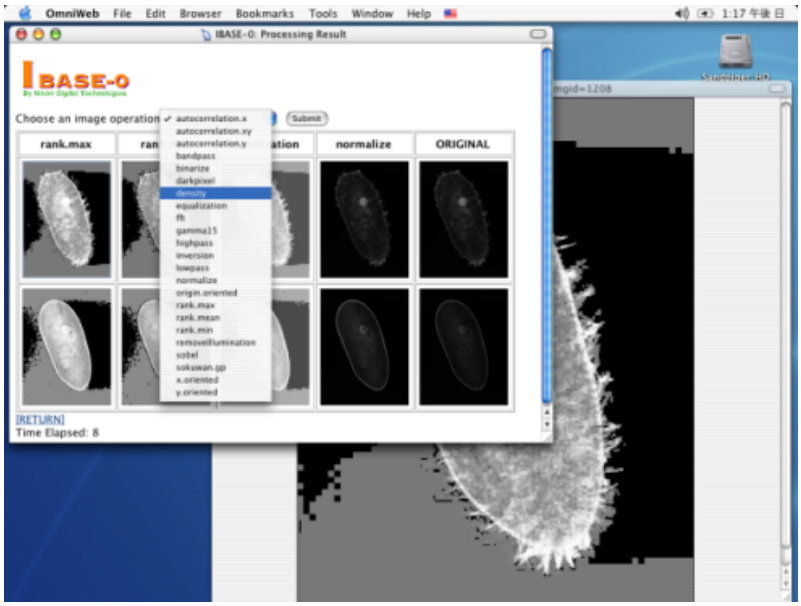


Figure 3: Applying filters sequentially in IBASE-0

Table 3: Response time and memory consumption

	Time [sec]	Memory [MB]
thresholding	21	32
automatic thresholding	29	41
gamma conversion	20	34
sobel	35	45
fft	39	50

* Measurements have done on Linux 2.4.13 / Pentium 933MHz / 512MB using 10 images which size is 480×640 pixels.

At last, we point out a few inconvenient points of the current R and R-surrounding environment. They are concerning (a) vector style programming, (b) plotting image, and (c) foreign interface.

Vector style programming is powerful but S functions sometimes seems to be ad hoc. For example, functions returning a scalar value are confusing for developers who are not familiar with the way of statistics. Function `mean` returns a scalar value for the both of a vector and a matrix. On the other hand, function `var` returns a scalar and a vector for a vector and a matrix respectively. The design of polymorphic function varies without clear criterion, that is, some are generic function but some others are not.

Plotting image is slow using function `plot`. When a repaint of window occurs, it is intolerably slow. As a temporary solution, we have developed an image display function using the Tcl/Tk interface.

Calling R functions from the outside of R is troublesome in the practical sense. The omega project provides several foreign interfaces[4]. However, version dependency between modules is so strong that we employed primitive but robust method, which is forking R.

5 Conclusion

We have developed rimage, an image processing library for R, and IBASE-0, an integrated database for image processing algorithm design. We conclude that R can be successfully integrated in practical development process in the image engineering field.

References

[1] J. Chambers. Evolution of the S language. In *Computing Science and Statistics*, Vol. 28. Interface Foundation of North America, Fairfax Station, Virginia, 1996.

[2] FFTW. http://www.fftw.org.

[3] Independent JPEG Group. http://www.ijg.org.

[4] The Omega Project. http://www.omegahat.org/.

[5] T. Takashina. How to use ILAS. http://homepage.mac.com/t_takashina/ILAS.

[6] T. Takashina, K. Tanaka, and S. Watanabe. Individual level analysis using decision making features in multiagent based simulation. In K Kuwabara and J Lee, editors, *5th Pacific Rim International Workshop on Multi-Agents, Tokyo, Japan, August 18-19, 2002. Proceedings*. Springer, 2002.