This manual documents Guile-CV version 0.4.0.

Copyright (C) 2016 - 2023 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License.”
Table of Contents

Preface .................................................. 1
  Contributors to this Manual ............................. 1
  Join the GNU Project .................................. 1
  The Guile-CV License ................................. 1

I. Introduction .......................................... 1
  About Guile-CV ........................................ 2
  Obtaining and Installing Guile-CV ...................... 2
  Contact Information .................................... 6
  Reporting Bugs .......................................... 6

II. Using Guile-CV ...................................... 6
  Configuring Guile for Guile-CV ......................... 6
    Configuring Guile's repl-print procedure ................. 6
    Configuring Guile's raised exception system .......... 7
  Images used in Guile-CV's documentation ............... 9
  Starting Guile-CV ..................................... 9

III. Guile-CV Core Reference ........................... 10
  Overview ............................................... 10
    Naming Conventions .................................. 10
    Abreviations ........................................ 10
  Image Processing ...................................... 10
    Image Structure and Accessors ......................... 10
    Kernel Structure and Accessors ....................... 13
    Import Export ...................................... 19
    Histogram .......................................... 20
  Texture ............................................... 21
  Features ............................................... 23
  Particles ............................................. 26
  Filter ................................................ 26
  Process .............................................. 29
  Transform ............................................ 32
  Morphology .......................................... 35
  Segmentation ......................................... 39
  Compose ............................................... 40
  Utilities .............................................. 41

Support ................................................ 43
  Modules ............................................... 44
  Goops ................................................. 44
  Pi ...................................................... 44
  Utils .................................................. 44
Appendix A  GNU Free Documentation License  44

Concept Index  ...........................................  53
Procedure Index .........................................  54
Variable Index ...........................................  56
Type Index ...............................................  57
Preface

This manual describes how to use Guile-CV. It relates particularly to Guile-CV version 0.4.0.

Contributors to this Manual

Like Guile-CV itself, the Guile-CV reference manual is a living entity. Right now, the contributor to this manual is:

• David Pirotte

who is also the author and maintainer of Guile-CV.

You are most welcome to join and help. Visit Guile-CV (http://www.gnu.org/software/guile-cv) web site to find out how to get involved.

Join the GNU Project

GNU Guile-CV is part of the GNU Operating System, developed by the GNU Project.

If you are the author of an awesome program and want to join us in writing Free (libre) Software, please consider making it an official GNU program and become a GNU Maintainer. You can find instructions on how to do this here (https://www.gnu.org/help/evaluation.html).

You don’t have a program to contribute? Look at all the other ways you may help (https://www.gnu.org/help/help.html).

To learn more about Free (libre) Software, you can read and please share this page (https://gnu.org/philosophy/free-sw.html).

The Guile-CV License

Guile-CV is Free Software. Guile-CV is copyrighted, not public domain, and there are restrictions on its distribution or redistribution:

• Guile-CV and supporting files are published under the terms of the GNU General Public License version 3 or later. See the file LICENSE.

• This manual is published under the terms of the GNU Free Documentation License (see Appendix A [GNU Free Documentation License], page 44).

You must be aware there is no warranty whatsoever for Guile-CV. This is described in full in the license.

I. Introduction
I. Introduction

About Guile-CV

GNU Guile-CV
Image Processing and Analysis in Guile (http://www.gnu.org/software/guile)
a Computer Vision functional programming library


Guile-CV (http://www.gnu.org/software/guile-cv) is based on Vigra (http://ukoethe.github.io/vigra/) (Vision with Generic Algorithms), enhanced with additional algorithms (Image Textures, Delineate, Reconstruction and many more), all accessible through a nice, clean and easy to use high level API.

Guile-CV (http://www.gnu.org/software/guile-cv) is natively multi-threaded, and takes advantage of multiple cores, using high-level and fine grained application-level parallelism constructs available in Guile (http://www.gnu.org/software/guile), based on its support to POSIX threads.

Objective

Guile-CV (http://www.gnu.org/software/guile-cv) objective is to be a robust, reliable and fast - Image Processing and Analysis - Computer Vision functional programming library for the Guile (http://www.gnu.org/software/guile) Scheme language. Guile-CV (http://www.gnu.org/software/guile-cv) also wants to be easy to use, study, modify and extend.

Guile-CV (http://www.gnu.org/software/guile-cv) can be used as an educational software, a research toolbox but it can also be used 'in production': Guile-CV (http://www.gnu.org/software/guile-cv) is robust, reliable and fast, and we will make sure Guile-CV (http://www.gnu.org/software/guile-cv) remains robust, reliable and fast as it grows.

Savannah


Obtaining and Installing Guile-CV

Guile-CV (http://www.gnu.org/software/guile-cv) can be obtained from the following archive site (http://ftp.gnu.org/gnu/guile-cv). The file will be named guile-cv-version.tar.gz. The current version is 0.4.0, so the file you should grab is:

http://ftp.gnu.org/gnu/guile-cv/guile-cv-0.4.0.tar.gz

Dependencies

Guile-CV (http://www.gnu.org/software/guile-cv) needs the following software to run:

- Autoconf >= 2.69
- Automake >= 1.14
- Makeinfo >= 6.6
I. Introduction

- Guile ([http://www.gnu.org/software/guile](http://www.gnu.org/software/guile)) >= 2.0.14  
  [allows 2.2, 3.0 (>= 3.0.7)]
- Guile-Lib ([http://www.nongnu.org/guile-lib](http://www.nongnu.org/guile-lib)) >= 0.2.5
- Vigra ([http://ukoethe.github.io/vigra/](http://ukoethe.github.io/vigra/)) >= 1.11.0

**Note:**
If you manually install Vigra ([http://ukoethe.github.io/vigra/](http://ukoethe.github.io/vigra/)), make sure you pass the cmake DCMAKE_BUILD_TYPE=RELEASE option, which triggers absolutely essential adequate runtime optimization flags.

- Vigra C ([https://github.com/BSepcke/vigra_c](https://github.com/BSepcke/vigra_c)) >= commit 0af647d08 - Oct 28, 2018

The local minima and maxima interfaces have been improved, and now support the full set of options provided by Vigra, to our request (thank you Benjamin!). In addition there has been a few bugs fixed, including one we detected while working on Guile-CV local minima bindings.

Vigra C - a C wrapper [to some of] the Vigra functionality - is currently only available by cloning its source code git ([https://git-scm.com/](https://git-scm.com/)) repository: there is no release and no versioning scheme either\(^1\). But no big deal, its home page has an 'Installation' section which guides you step by step.

**Notes:**
1. Make sure you pass the cmake DCMAKE_BUILD_TYPE=RELEASE option, which triggers absolutely essential adequate runtime optimization flags;
2. Vigra C says it depends on cmake >= 3.1, but this is only true if you want to build its documentation, probably not the case. Most distribution still have cmake 2.8, if that is your case, you may safely edit /your/path/vigra_c/CMakeLists.txt and downgrade this requirement to the cmake version installed on your machine;
3. Make sure the directory where libvigra_c.so has been installed is 'known', either because it is defined in /etc/ld.so.conf.d, or you set the environment variable LD_LIBRARY_PATH, otherwise Guile won't find it and configure will report an error.

- LaTeX ([http://www.latex-project.org/](http://www.latex-project.org/))

Any modern latex distribution will do, we use TexLive ([https://tug.org/texlive/](https://tug.org/texlive/)). Guile-CV will check that it can find the standalone documentclass, as well as the following packages: inputenc, fontenc, lmodern, xcolor, booktabs, siunitx, iwona.

Iwona ([http://www.tug.dk/FontCatalogue/iwona/](http://www.tug.dk/FontCatalogue/iwona/)): this is the font used to create [im-histogram], page 20, headers, legend indices and footers. Note that it could be that it is not part of your 'basic' LaTex distro, on debian for example, iwona is part of the texlive-fonts-extra package.

---

\(^1\) We do our best to check that the libvigra_c installed library does contain the required Guile-CV functionality though, and these checks are listed as part of our configure steps.
Install from the tarball

Assuming you have satisfied the dependencies, open a terminal and proceed with the following steps:

```
  cd <download-path>
tar zxf guile-cv-0.4.0.tar.gz
cd guile-cv-0.4.0
./configure [--prefix=/your/prefix] [--with-guile-site=yes]
make
make install
```

Special note:

Before you start to use Guile-CV (http://www.gnu.org/software/guile-cv), make sure you read and implement the recommendation made in the manual, section See [Configuring Guile for Guile-CV], page 6.

Happy Guile-CV (http://www.gnu.org/software/guile-cv)!

Install from the source


There are currently 2 [important] branches: master and devel. Guile-CV (http://www.gnu.org/software/guile-cv) stable branch is master, developments occur on the devel branch.

So, to grab, compile and install from the source, open a terminal and:

```
  git clone git://git.savannah.gnu.org/guile-cv.git
cd guile-cv
./autogen.sh
./configure [--prefix=/your/prefix] [--with-guile-site=yes]
make
make install
```

Special note:

Before you start to use Guile-CV (http://www.gnu.org/software/guile-cv), make sure you read and implement the recommendation made in the manual, section See [Configuring Guile for Guile-CV], page 6.

The above steps ensure you're using Guile-CV (http://www.gnu.org/software/guile-cv) bleeding edge stable version. If you wish to participate to developments, checkout the devel branch:

```
  git checkout devel
```

Happy hacking!

Notes:

1. The default and --prefix installation locations for source modules and compiled files (in the absence of --with-guile-site=yes) are:
I. Introduction

$(datadir)/guile-cv
$(libdir)/guile-cv/guile/$(GUILE_EFFECTIVE_VERSION)/site-ccache

If you pass --with-guile-site=yes, these locations become the Guile global site and site-ccache directories, respectively.

The configure step reports these locations as the content of the sitedir and siteccachedir variables, respectively the source modules and compiled files install locations. After installation, you may consult these variables using pkg-config:

    pkg-config guile-cv-1.0 --variable=sitedir
    pkg-config guile-cv-1.0 --variable=siteccachedir

You will need - unless you have used --with-guile-site=yes, or unless these locations are already 'known' by Guile - to define or augment your GUILE_LOAD_PATH and GUILE_COMPILED_PATH environment variables with these locations, respectively (or %load-path and %load-compiled-path at run time if you prefer\(^2\) (See Environment Variables (https://www.gnu.org/software/guile/manual/guile.html#Environment-Variables) and Load Path (https://www.gnu.org/software/guile/manual/guile.html#Load-Paths) in the Guile Reference Manual).

2. Guile-CV also installs its libguile-cv.* library files, in $(libdir). The configure step reports its location as the content of the libdir variable, which depends on on the content of the prefix and exec_prefix variables (also reported). After installation, you may consult these variables using pkg-config:

    pkg-config guile-cv-1.0 --variable=prefix
    pkg-config guile-cv-1.0 --variable=exec_prefix
    pkg-config guile-cv-1.0 --variable=libdir

You will need - unless the $(libdir) location is already 'known' by your system - to either define or augment your $LD_LIBRARY_PATH environment variable, or alter the /etc/ld.so.conf (or add a file in /etc/ld.so.conf.d) and run (as root) ldconfig, so that Guile-CV finds its libguile-cv.* library files\(^3\).

3. To install Guile-CV, you must have write permissions to the default or $(prefix) directory and its subdirs, as well as to both Guile's site and site-ccache directories if --with-guile-site=yes was passed.

4. Like for any other GNU Tool Chain compatible software, you may install the documentation locally using make install-info, make install-html and/or make install-pdf.

5. Last but not least :), Guile-CV comes with a test-suite, which we recommend you to run (especially before [Reporting Bugs], page 6):

    make check

\(^2\) In this case, you may as well decide to either alter your $HOME/.guile personal file, or, if you are working in a multi-user environment, you may also opt for a global configuration. In this case, the file must be named init.scm and placed it here (evaluate the following expression in a terminal): guile -c "(display (%global-site-dir))(newline)".

\(^3\) Contact your administrator if you opt for the second solution but don’t have write privileges on your system.
II. Using Guile-CV

Contact Information

Mailing lists
Guile-CV uses the following mailing list:

- **guile-user@gnu.org** is for general user help and discussion.
- **guile-devel@gnu.org** is used to discuss most aspects of Guile-CV, including development and enhancement requests.

Please use ‘Guile-CV - ’ to proceed the subject line of Guile-CV related emails, thanks!

You can (un)subscribe to the one or both of these mailing lists by following instructions on their respective list information page (https://lists.gnu.org/mailman/listinfo/).

IRC
Most of the time you can find me on irc, channel #guile, #guix and #scheme on irc.libera.chat, #introspection, #gtk and #clutter on irc.gnome.org, under the nickname daviid.

Reporting Bugs
Guile-CV uses the following bug reports mailing list:

- **bugs-guile-cv@gnu.org**

You can (un)subscribe to the bugs report list by following instructions on the list information page (https://lists.gnu.org/mailman/listinfo/bug-guile-cv).

Further information and a list of available commands are available here (https://debbugs.gnu.org/server-control.html).

II. Using Guile-CV

Configuring Guile for Guile-CV
Guile must be modified, with respect to two core functionalities, before to start to use Guile-CV: (a) its repl-print procedure and (b) its raised exception system.

Configuring Guile’s repl-print procedure
Guile’s repl-print procedure calls (write val), which is inadequate for Guile-CV images - or for that matter, for any work that involves very large data structure manipulations - even very small images 4. Fortunately, Guile provides both a simple mechanism to alter the default repl printer and the alternate repl printer procedure we need: truncated-print.

---

4 Even for very small images, using write is inadequate, in a terminal, and will definitely kill your Emacs/Geiser session. Not to mention it will raise your electricity bill :) - till you succeed to delete its process, Emacs will use one core at more then 100%, desperately trying to display hundreds of thousands of floating point values, heating your laptop (if you have a laptop) up to the point you’ll be able to cook an egg on it, and get its fans crazy... You’ve been warned :).
II. Using Guile-CV

To modify the default repl printer, you may alter (or add if it doesn’t exist) your $HOME/.guile file or, if you are working in a multi-user environment, you may alter (or add if it doesn’t exist) the file named init.scm in the Guile global site directory.

Which ever solution you choose, add the following lines:

```scheme
(use-modules (ice-9 pretty-print)
             (system repl common))

(repl-default-option-set! 'print
   (lambda (repl obj)
      (truncated-print obj) (newline)))
```

Configuring Guile’s raised exception system

Guile’s core raised exception printers call simple-format, which is inadequate for Guile-CV images - or for that matter, for any work that involves very large data structure manipulations - even very small images (see the related footnote of the previous section, it explains how ‘inadequate’ this default is for Guile-CV images).

Unfortunately, Guile does not provide an easy mechanism to alter its core raised exception printers. This leaves us with no other option but making some changes to the module where those are defined, namely the (ice-9 boot-9) Guile’s core module, which then needs to be (re)compiled and (re)installed.

As the (ice-9 boot-9) Guile’s core module has changed from 2.0, 2.2 to 3.0, and still is subject to change any time in the future, we can’t provide a ‘one patch for all’ solution.

Instead, we describe the steps to manually update your local version. However if you think it is ‘too much’ for you, get in touch with us, and we will guide you or provide a ‘ready to use module’, depending on your version of Guile.

So, let’s first figure out where the (ice-9 boot-9) resides on your system, in a guile session, enter the following:

```scheme
(string-append (%package-data-dir) "" (effective-version))
⇒
$2 = "/opt3/share/guile/3.0"
```

The above returned value is an example of course, just proceed with the value returned by your system. So, the file we need to edit, in our example, is here:

```
/opt3/share/guile/3.0/ice-9/boot-9.scm
```

Edit the above file and:

1. Search for the line `(define format simple-format)`, and below, add a line containing `(define exception-format simple-format)`, so now your version of the file looks like this:

---

5 The Guile global site directory location may be obtained by evaluating the following expression in a terminal: `guile -c "(display (%global-site-dir))(newline)"`. You need write privileges to add or modify this file, contact your system administrator if you’re not in charge of the system you are working on.

6 Special thanks to Daniel Llorens, who proposed these changes, without which it would just be impossible to work with Guile-CV - or for that matter, any work that involves very large data structure manipulations.

7 You need write privileges to modify this module, contact your admin if you’re not in charge of the system you are working on.
II. Using Guile-CV

(define format simple-format)
(define exception-format simple-format)

2. Replace all occurrences of 
(format' using '(exception-format') [note and meticulously respect the presence of the leading open paren '(' and the trailing space ')' in both the search and replace expressions].

Save the file.

3. Compile the file - in the following lines, substitute /opt3 by your $prefix value, 3.0 by your guile (effective-version) as well as $HOME:
   cd /opt3/share/guile/3.0/ice-9
   guild compile boot-9.scm
   ⊣
   ;; note: source file /opt3/share/guile/3.0/ice-9/boot-9.scm
   ;; newer than compiled /opt3/lib/guile/3.0/ccache/ice-9/boot-9.go

   Note that the target (compiled) filename is boot-9.scm.go - not boot-9.go.

4. Install the compiled file:
   /opt3/lib/guile/3.0/ccache/ice-9/boot-9.go

   Finally, once the above is completed, add the following lines8 to your $HOME/.guile or, if you are working in a multi-user environment, to the file named init.scm in the so-called Guile global site directory (the previous subsection lists the terminal command you need to run to see where that directory is on your system):

   (define %n-char-limit 400)
   (define %n-char-limit-fmt-expr
     (simple-format #f "~~~a@y" %n-char-limit))

   (define (rewrite-fmt fmt tell)
     (let loop ((f "")
                  (b 0))
       (let ((next (string-contains-ci fmt tell b)))
         (if next
             (loop (if (or (zero? next) (not (char=? #\~ (string-ref fmt (- next 1)))))
                   (string-append f
                                   (substring fmt b next)
                                   %n-char-limit-fmt-expr)
                   f)
                   (+ next 2))
             (string-append f (substring fmt b)))))

   (when (defined? 'exception-format)

---

8 Early versions of Guile-CV used to recommend an exception-format setting based on truncated-print, which works as expected if you are using Guile 2.0 or 2.2, but using Guile 3.0, a raised exception would lead to a series of 'Unwind-only stack overflow exception' and exit Guile abruptly.
II. Using Guile-CV

(set! exception-format
  (lambda (port fmt0 . args)
    (apply (@ (ice-9 format) format)
      port
      (rewrite-fmt (rewrite-fmt fmt0 "-s") "-a")
      args))))

Feel free to adapt the %n-char-limit value to your own taste.

You are now ready to use Guile-CV!

Images used in Guile-CV’s documentation

Images used in Guile-CV’s documentation are distributed with the source and installed here:

$prefix/share/doc/guile-cv/images

Examples using im-load and im-save given in this manual, unless a full pathname is specified, assume that these images are available from the guile current working directory, see getcwd and chdir in Guile's manual

Our best recommendation, at least to start with, is to create a working directory, such as mkdir $HOME/guile-cv/images, for example, and as you need them, copy the distributed images you are interested in.

Starting Guile-CV

- SPECIAL NOTE -

Before you start to use Guile-CV, make sure you read and implement the recommendation made in [Configuring Guile for Guile-CV], page 6,

With the previous [Images used in Guile-CV’s documentation], page 9, recommendations in mind, open a terminal and:

```
cd ~/guile-cv/images
guile
scheme@(guile-user)> ,use (cv)
scheme@(guile-user)> (im-load "sand.tif")
⇒
  $2 = (512 512 1 (#f32(125.0 128.0 124.0 118.0 108.0 75.0 76.0 # ...)))
```

Or if you use Emacs (https://www.gnu.org/software/emacs) which, coupled with Geiser (http://www.nongnu.org/geiser) absolutely rocks :-), then a typical session becomes:

```
fire Emacs
M-x cd
-|
Change default directory: ~/guile-cv/images

M-x run-guile
scheme@(guile-user)> ,use (cv)
```
III. Guile-CV Core Reference

Overview

FIXME - The overview section and its subsections is a mock-up, all need to be actually 'filled'.

Naming Conventions

Vigra Funtions

Guile-CV low level CR procedure names that bind a Vigra functions always start with 
vigra-

    vigra-local-minima
    vigra-crop-channel

...  

Abreviations

FIXME. Needs to be 'filled'.

Image Processing

Image Structure and Accessors

The Guile-CV procedures and methods related to image data structure, creating, accessing and copying images.

Procedures

Description

A Guile-CV image is represented by a list containing the following elements:

    (width height n-channel idata)

where idata is a list of n-channel elements, each element being a vector of (* width height) cells. More precisely, each element is an srfi-4 homogeneous numeric vector of 32 bit floats, called f32vector, knowing that f32 is the C type float.

The external representation (ie. read syntax) for idata vectors is #f32(...). As an example, a gray scale image of width 3 and height 2, initialized to 0.0 is represented by the following expression:

    (3 2 1 (#f32(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0)))
The \textit{n-channel} is an integer $\geq 1$, with no limit but the memory size. This said, most Guile-CV procedures and methods expect either GRAY scale ($n\text{-channel}=1$), or RGB ($n\text{-channel}=3$) images. For the later, the channels are \texttt{Red, Green and Blue} in that order.

Guile-CV provides useful accessors for all these fields. However, very often, you will need them all, in which case your best friend is (\texttt{ice-9 match}), here is an example:

```
(use (cv))
(define image (im-make 4 3 3))
(match image
  ((width height n-chan idata)
   (match idata
     ((r g b)
      ... your code here ...))))
```

You will find many examples of such a ‘pattern’ in Guile-CV’s source code itself of course, along with some other ‘techniques’ that might be useful, so we invite you to read it, and if you do so: feedback, design and code review is more then welcome! This section describes what is in the module (\texttt{cv idata}).

Note that the (\texttt{cv}) module imports and re-exports, among may others, the public interface of (\texttt{ice-9 match}).

\section*{Procedures}

\begin{itemize}
  \item \texttt{im-make width height n [value]} \hspace{1em} \texttt{[Procedure]}
  \item \texttt{im-make-channel width height [value]} \hspace{1em} \texttt{[Procedure]}
  \item \texttt{im-make-channels width height n [value]} \hspace{1em} \texttt{[Procedure]}
    \begin{verbatim}
    Returns a new image, list of channels or channel.
    Each channel is an srfi-4 homogeneous vector of 32 bit floats (f32vector),
    of \texttt{width} by \texttt{height} initialized to \texttt{value}. The default \texttt{value} is 0.0
    \end{verbatim}
  \item \texttt{im-copy image} \hspace{1em} \texttt{[Procedure]}
  \item \texttt{im-copy-channel channel width height} \hspace{1em} \texttt{[Procedure]}
    \begin{verbatim}
    Returns a new fresh copy of \texttt{image} or \texttt{channel}.
    \end{verbatim}
  \item \texttt{im-size image} \hspace{1em} \texttt{[Method]}
    \begin{verbatim}
    Returns the list of (width height n-channel)for \texttt{image}.
    \end{verbatim}
  \item \texttt{im-width image} \hspace{1em} \texttt{[Method]}
  \item \texttt{im-height image} \hspace{1em} \texttt{[Method]}
  \item \texttt{im-n-channel image} \hspace{1em} \texttt{[Method]}
  \item \texttt{im-channels image} \hspace{1em} \texttt{[Method]}
  \item \texttt{im-channel image n} \hspace{1em} \texttt{[Procedure]}
    \begin{verbatim}
    Returns, respectively the width, the height, n-channel, channels or the nth channel
    for \texttt{image}.
    \end{verbatim}
  \item \texttt{im-image? image} \hspace{1em} \texttt{[Procedure]}
  \item \texttt{im-gray? image} \hspace{1em} \texttt{[Method]}
\end{itemize}
### Method

**im-rgb? image**

Returns #t if image is respectively a Guile-CV image, a GRAY scale or an RGB image.

**im-binary? i1 i2 i3 ...**  
**im-binary-channel? width height c1 c2 c3 ...**

Returns #t if i1 i2 i3 ... or c1 c2 c3 ... respectively are BINARY (Black and White) images or channels respectively.

Note that when more than one image or channel is passed, they must all be of the same size.

**im-=? [precision] i1 i2 i3 ...**  
**im-=channel? width height [precision] c1 c2 c3 ...**

Returns #t if i1 i2 i3 ... or c1 c2 c3 ... respectively are of the same size, have the same number of channels that all respectively contain the same values.

If the first argument is a number, it is used as the precision to compare pixel values. The default precision value is 1.0e-4. Note that if you are certain your images or channels contain 'discrete' float values, you may pass 0.0 as the precision to be used, in which case values will be compared using = (instead of float=?), which is faster.

**im-ref image i j [k]**  
**im-fast-ref image i j [k]**

Returns the pixel value stored at position i and j of the image channel k. The default value for k is 0.

**im-fast-ref** does not check the validity of its arguments: use it at your own risk.

**im-set! image i j [k] value**  
**im-fast-set! image i j [k] value**

Returns nothing.

Sets the pixel value stored at position i and j of the image channel k to value. The default value for k is 0.

**im-fast-set!** does not check the validity of its arguments: use it at your own risk.

**im-channel-offset i j width height**  
**im-fast-channel-offset i j width**

Returns the channel offset for the i and j indices, based on the width and height of the channel.

This procedure converts the matrix indices i and j to a vector offset for a channel of size width and height.

**im-fast-channel-offset** does not check the validity of its arguments: use it at your own risk.

**im-channel-ref channel i j width height**  
**im-fast-channel-ref channel i j width**

Returns the pixel value stored at position i and j of the channel of size width and height.
im-fast-channel-ref does not check the validity of its arguments: use it at your own risk.

**im-channel-set! channel i j width height value**

**im-fast-channel-set! channel i j width value**

 Returns nothing.

Sets the pixel at position $i$ and $j$ of channel of size $width$ and $height$ to $value$.

im-fast-channel-set! does not check the validity of its arguments: use it at your own risk.

**im-collect what i1 i2 i3 . . .**

 Returns a list of $what$ collected from $i1 i2 i3 . . .$

The valid $what$ symbols are:

- size
- width
- height
- n-channel
- channels
  - chan-0, gray, red
  - chan-1, green
  - chan-2, blue
  - chan-k (*)

(*): whith $k$ being a valid channel indice, $[0 (- n 1)]$.

**Kernel Structure and Accessors**

The Guile-CV procedures and methods related to kernel data structure, creating and accessing kernels.

**Procedures**

**Description**

A Guile-CV kernel ([https://en.wikipedia.org/wiki/Kernel_(image_processing)](https://en.wikipedia.org/wiki/Kernel_(image_processing))) is represented by a list containing the following elements:

$$(width \ height \ kdata)$$

where $kdata$ is a vector of $(* width height)$ cells. More precisely, $kdata$ is an srfi-4 homogeneous numeric vector of 64 bit floats, called f64vector, knowing that f64 is the C type double.

The external representation (ie. read syntax) for $kdata$ vectors is #f64(...). As an example, the identity kernel of width 3 and height 3, initialized to 0.0 is represented by the following expression:

$$((3 3 \ #f64(0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 0.0))$$

The kernel $width$ and $height$ can be different (kernels can be rectangular), but both $width$ and $height$ must be odd values.
Guile-CV provides useful accessors for kernel fields, however, if you need them all, just like for accessing image fields, your best friend is (ice-9 match), here is an example:

```
(use (cv)
(match kernel
  ((width height kdata)
   ... your code here ...))
```

Note that the (cv) module imports and re-exports, among many others, the public interface of (ice-9 match).

Guile-CV defines a few useful kernels, see [kernel variables], page 17, at the end of this section, that you both may want to use and reuse: it will be easier, if you need to do so, to define your own kernels reusing an existing one, see the (cv kdata) module.

**Procedures**

**k-make** width height [values #f] [norm #f]  [Procedure]

Returns a new kernel.

The `kdata` value of this new kernel is an srfi-4 homogeneous numeric vector of 64 bit floats, `f64vector`, composed of width by height cells.

The optional *values* argument can be:

- **#f** `kdata` is initialized to the ‘identity’ kernel (all zeros except the center of the kernel, initialized to 1)
- a single value
  all `kdata` cells are initialized using that single value
- a list of values
  a list of width by height values, used to initialize `kdata`, in the order they are given

The optional *norm* argument can be:

- **#f** in this case, `kdata` is not normalized
- **#t** unless *values* would be #f, `kdata` is normalized using (reduce + 0 values)
- a single value
  all `kdata` cells are normalized using that value, which must be a number different from 0

When both *values* and *norm* are passed - which is mandatory if you want to pass *norm* (since these are optional arguments, as opposed to keyword arguments) - *values* must precede *norm* on the arguments list.

As an example, here is how to define a 3 x 3 normalized mean kernel:

```
(use (cv)
  (k-make 3 3 1 #t)
  │
  $2 = (3 3 #f64(0.1111111111111111 0.1111111111111111 # # # ...))
```
(k-display $2)

-[

0.11111 0.11111 0.11111
0.11111 0.11111 0.11111
0.11111 0.11111 0.11111

k-make-circular-mask radius [value 1] [norm #f] [Procedure]

Returns a new circular mask kernel.

The kdata value of this new kernel is an srfi-4 homogeneous numeric vector of 64 bit floats, f64vector, composed of width by height cells where width and height are equal and odd values determined by the procedure.

The mandatory radius argument must be a floating point number satisfying the following predicate: (float>=? radius 0.5).

The optional norm argument can be:

#f in this case, kdata is not normalized
#t kdata values are normalized using (* n value), where n is the number of non zero elements of the circular kernel mask being defined.

When both value and norm are passed - which is mandatory if you want to pass norm (since these are optional arguments, as opposed to optional keyword arguments) - value must precede norm on the arguments list.

To illustrate, here are the circular kernel masks of radius 0.5, 1, 1.5 respectively:

... (for-each (lambda (i)
  (k-display (k-make-circular-mask i)
   #:proc float->int))
  '(0.5 1.0 1.5))
-[

0 1 0
1 1 1
0 1 0

1 1 1
1 1 1
1 1 1

0 0 1 0 0
0 1 1 1 0
1 1 1 1 1
0 1 1 1 0
0 0 1 0 0
To better illustrate, let’s define a bigger circular kernel mask, transform it to an image and [im-show], page 42, it:

```
... (match (k-make-circular-mask 49)
   ((w h kdata) (list w h 1 (list (f64vector->f32vector kdata)))))
⊣ $6 = (99 99 1 (#f32(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 # ...))))
   (im-show $6 'scale)
```

And you should see the following image

9

<table>
<thead>
<tr>
<th>Procedure</th>
<th>k-width kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure</td>
<td>k-height kernel</td>
</tr>
<tr>
<td>Procedure</td>
<td>k-size kernel</td>
</tr>
<tr>
<td>Procedure</td>
<td>k-channel kernel</td>
</tr>
</tbody>
</table>

Returns, respectively, the width, the height, the list of width and height or the kdata for kernel.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>kernel? kernel</th>
</tr>
</thead>
</table>

Returns #t if kernel is a Guile-CV kernel.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>k-ref kernel i j</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure</td>
<td>k-fast-ref kernel i j</td>
</tr>
</tbody>
</table>

Returns the value stored at position i and j of the kernel.

k-fast-ref does not check the validity of its arguments: use it at your own risk.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>k-set! kernel i j value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure</td>
<td>k-fast-set! kernel i j value</td>
</tr>
</tbody>
</table>

Sets the value stored at position i and j of the kernel to value.

k-fast-set! does not check the validity of its arguments: use it at your own risk.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>k-offset i j width height</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedure</td>
<td>k-fast-offset i j width</td>
</tr>
</tbody>
</table>

Returns the kernel offset for the i and j indices, based on the width and height of the kernel.

This procedure converts the matrix indices i and j to a vector offset for a kernel of size width and height.

k-fast-offset does not check the validity of its arguments: use it at your own risk.

<table>
<thead>
<tr>
<th>Procedure</th>
<th>k-display image [#:proc #:f] [#:port (current-output-port)]</th>
</tr>
</thead>
</table>

Returns nothing.

Displays the content of kernel on port, applying proc to each kernel value.

---

9 The 'scale optional argument passed to [im-show], page 42, as its name indicate, is so that kernel values will be scaled, which in this case means that 1.0 values will become 255.0 - otherwise, it would be almost impossible for a human eye to actually see the shape of the circle ...
Variables

Notes: (a) the following kernels are merely offered as ‘didactic’ examples, some of these were used ‘in the old days’, but in most cases, you will find and prefer to use a ‘specific’ and ‘modern’ procedure that will give (much) better results, such as, [im-gaussian-blur], page 26, [im-gaussian-sharp], page 27, [im-sharpen], page 27, (a simple sharpening procedure), [im-canny], page 39, (edge detection) ... and (b) in order to make these definitions easier to read, we’ve added some spaces and newlines.

%k-identity

(k-display %k-identity #:proc float->int)
-|-0 0 0
   0 1 0
   0 0 0

%k-edge0

(k-make 3 3
  '( 1 0 -1
     0 0 0
     -1 0 1 ))

%k-edge1

(k-make 3 3
  '( 0 1 0
     1 -4 1
     0 1 0 ))

%k-sharpen

(k-make 3 3
  '( -1 -1 -1
     -1 9 -1
     -1 -1 -1 ))

%k-mean

(k-make 3 3
  '( 1 1 1
     1 1 1
     1 1 1 )
  9)

%k-gaussian-blur0

(k-make 3 3
  '( 0.37500 0.25000 0.37500
     0.25000 -2.50000 0.25000
     0.37500 0.25000 0.37500 )
-|0.25000 0.37500 0.25000
  0.25000 -2.50000 0.25000
  0.37500 0.25000 0.37500
(k-display %k-laplacian)
'( 1 2 1
    2 4 2
    1 2 1 )

16)

%k-gaussian-blur1

   (k-make 5 5
     '( 1 4 6 4 1
       4 16 24 16 4
       6 24 36 24 6
       4 16 24 16 4
       1 4 6 4 1 )
   )

256)

%k-unsharp

   (k-make 5 5
     '( 1 4 6 4 1
       4 16 24 16 4
       6 24 -476 24 6
       4 16 24 16 4
       1 4 6 4 1 )
   )

-256)

%k-emboss

   Also called %k-compass or %k-directional, this kind of filter is useful to enhance edges in given directions. With a 3 x 3 kernel, one normally uses filters for 0, 45, 90 and 135 degrees. The various angles are obtained ‘rotating’ the positive and negative values to ‘align’ with the various directions.

   (k-make 3 3
     '( -2 -2 0
       -2 6 0
       0 0 0 ))

%k-laplacian

   This is a variation of the more traditional Laplacian kernels, that are meant to enhance edges, in this case in an isotropic fashion (non-directional). This the implementation in the Vigra code and it attributes large weights to the diagonal pixels of the kernel. Nevertheless, the total weight is zero.

   (k-make 3 3
     '( 0.375 0.25 0.375
       0.25 -2.5 0.25
       0.375 0.25 0.375 ))

Prewitt filtering

%k-prewitt-y

   A 3 x 3 kernel which emphasizes horizontal edges by approximating a vertical gradient.

   (k-make 3 3
\begin{verbatim}
\%(k-prewitt-x)\nA 3 x 3 kernel which emphasizes vertical edges by approximating an horizontal gra-
dient.
\%(k-make 3 3
  \%(k-prewitt-x)
\)

\%(k-sobel-y)\nSobel filtering
Filtering an image using a `Sobel filter` requires a three steps approach: (1) filtering the
image using the `Sobel y filter`, (2) ditto using the `Sobel x filter`, and (3) combining
the results to obtain `Sobel magnitude`: \( \sqrt{\sqrt{\text{sobel-y}} + \sqrt{\text{sobel-x}}} \).
\%(k-make 3 3
  \%(k-sobel-y)
\)
\%(k-sobel-x)\nImport Export
The Guile-CV procedures and methods to load, save and query file system images.

Procedures
\%im-load\n\%(im-load filename)
Returns a Guile-CV image.
Loads the image pointed by \texttt{filename} and returns a Guile-CV image. \texttt{filename} can
either be a GRAY or an RGB image.
At this point, Guile-CV supports the following file formats: GIF, TIFF, JPEG, BMP,
EXR, HDR, PNM (PBM, PGM, PPM), PNG, SunRaster, Khoros-VIFF.
\%im-save\n\%(im-save image filename \[scale \#f\])
Returns \#t.
Saves \texttt{image} in \texttt{filename}.
The optional \texttt{scale} argument can take the following values:
\#f \hspace{1cm} pixel values are `clipped': values < 0 are saved as 0, values > 255
are saved as 255, and otherwise are saved unchanged
\end{verbatim}
#t all pixel values are scaled\textsuperscript{10} to the \([0 \ 255]\) range

The type in which \(\text{image}\) is saved is determined by the \(\text{filename}\) extension, as in the following example:

\[
\text{(im-load "edx.png")}
\]

\[
\text{(im-save $4 "/tmp/edx.jpg")}
\]

**im-size filename**

- Returns the list of \((\text{width} \ \text{height} \ \text{n-channel})\) for \(\text{filename}\).

**im-width filename**

- Returns, respectively the \text{width}, the \text{height} and the \text{n-channel} for \(\text{filename}\).

**im-height filename**

- Returns, respectively the \text{width}, the \text{height} and the \text{n-channel} for \(\text{filename}\).

**im-n-channel filename**

- Returns, respectively the \text{width}, the \text{height} and the \text{n-channel} for \(\text{filename}\).

**im-gray? filename**

- Returns \#t if \(\text{filename}\) is respectively a GRAY scale or an RGB image.

**im-rgb? filename**

- Returns \#t if \(\text{filename}\) is respectively a GRAY scale or an RGB image.

### Histogram

Other Guile-CV histogram procedures and methods.

### Procedures

**im-histogram image [#:subtitle "Untitled"]**

- Returns two values: (1) an image; (2) a list (or a list of list) of significant values for \(\text{image}\): one list if \(\text{image}\) is GRAY, a list of list of values per channel if \(\text{image}\) is RGB.

The returned image is composed of a header (title, #:subtitle), either the GRAY or the RGB channel histogram(s) for \(\text{image}\) and a footer, which is a table containing, for each channel, the following values: \text{mean}, \text{standard deviation}, \text{minimum}, \text{maximum}, the \text{mode}\textsuperscript{11} followed by its value.

Here below, the call sequence and the histogram for the GRAY image \text{sinter.png} given along with Guile-CV documentation and examples:

\[
\text{scheme@(guile-user)> (im-load "sinter.png")}
\]

\[
\text{$32 = (212 \ 128 \ 1 \ (#f32(25.0 \ 39.0 \ 50.0 \ 51.0 \ 45.0 \ # \ ...)))}
\]

\[
\text{scheme@(guile-user)> (im-histogram $32 #:subtitle "sinter.png")}
\]

\[
\text{$34 = (282 \ 271 \ 1 \ (#f32(255.0 \ 255.0 \ 255.0 \ 255.0 \ 255.0 \ # \ ...)))}
\]

\[
\text{$35 = (27136 \ 163.346 \ 75.081 \ 0 \ 243 \ 215 \ 727)}
\]

Note that histogram images returned by \text{im-histogram} have no borders, the above histogram has been padded - using \text{(im-padd $34 1 \ 1 \ 1 \ #:color '(96 96 96))} - for better readability, otherwise the title above and the table below would look as if they were not centered.

\textsuperscript{10} Note that in this particular context, \text{scale} does not mean a change in dimension, but rather bringing pixel values from the range they occupy in memory to the \([0 \ 255]\) range.

\textsuperscript{11} The \text{mode} is the integer corresponding to the histogram entry that received the maximum of hits, and the value displayed in parens precisely is the number of hits.
Texture
The Guile-CV procedures and methods related to image texture measures.

Procedures

Description
Although introduced a long time ago\textsuperscript{12}, image texture measures are still very actual, with new research and practice applications in many areas, as described in this (highly recommended) document\textsuperscript{13}.

Image texture measures are ‘\textit{descriptive statistics}’, derived from the ‘\textit{Gray Level Co-occurrence Matrices (GLCM)}’ and its associated set of ‘\textit{Gray Level Co-occurrence Probability (GLCP)}’ matrices.

Guile-CV GLCM and GLCP data structures are identical to the one used for Guile-CV images (See [Image Structure and Accessors], page 10). Although they are not images ‘\textit{per se}’, they are composed of four square matrices (four channels), of size n-gl (the number of gray levels to consider), and upon which we (and users) need to run linear algebra procedures, already defined and available in Guile-CV.

Guile-CV offers the 11th first \textit{texture measures}, out of the 14th originally proposed by Haralick et al., which are the most commonly used and adopted ones.

This reference manual assumes you are familiar with the concepts, terminology and mathematical formulas involved in the calculations of GLCMs, GLCPs and \textit{image texture measures}. If that is not the case, consider carefully reading one or both of the documents cited above (or any other tutorial or reference material of your choice of course).

Procedures

\begin{verbatim}
im-texture image n-gl [#:dist 1] [#:p-max 255] [#:use-log2 #f] [#:no-px-y0 #f] 
[Procedure]
\end{verbatim}

Returns a list.

The procedure calls [im-glcpc], page 22, passing \textit{image}, \textit{n-gl} (the number of gray levels to consider), \textit{dist} (the distance between the ‘\textit{reference}’ and the ‘\textit{neighbour}’ pixels) and \textit{p-max} (the \textit{image} (pixel) maximum value), then computes and returns a list of the 11th first texture measures proposed by Haralick et al., which are:

\footnotesize
\begin{itemize}
\item
\end{itemize}

\begin{small}
\textsuperscript{13} M. Hall-Beyer, GLCM Texture: A Tutorial v. 3.0 March 2017
\end{small}
III. Guile-CV Core Reference

- (h1) uniformity (angular second moment)
- (h2) contrast
- (h3) correlation
- (h4) variance (sum of squares)
- (h5) homogeneity (inverse difference moment)
- (h6) sum average
- (h7) sum variance
- (h8) sum entropy
- (h9) entropy
- (h10) difference variance
- (h11) difference entropy

The #:use-log2 optional keyword argument, which defaults to #f, is passed to the internal procedures that calculate the parameters h8, h9 and h11. The original formulas proposed by Haralck and al. use log, but I have seen a couple of implementations using log2.\(^\text{14}\)

The #:no-px-y0 optional keyword argument, which defaults to #f, is passed to the internal procedure that calculate the parameter h10. For some obscure reason, and only with respect to this parameter, I have seen some implementations eliminating the first element of the so-called Px-y, an intermediate f32vector result, which holds, as its first element, the sum of the elements of the main diagonal of the GLCP.\(^\text{15}\)

**Procedure**

im-glcp image n-gl [#:dist 1] [#:p-max 255]

Returns the GLCP for image.

The procedure calls [im-glcm], page 22, passing image, n-gl (the number of gray levels to consider), dist (the distance between the ‘reference’ and the ‘neighbour’ pixels) and p-max (the image (pixel) maximum value), adds GLCM’ (the transposed version of GLCM, so the result is symmetrical around the diagonal), then computes and returns the GLCP.

The returned GLCP is an ‘image’ composed four channels (four square matrices of size n-gl), corresponding to the (symmetrical) Gray Level Co-occurrences expressed as probabilities, each calculated at a specific ‘angle’, respectively 0°, 45°, 90°, and 135°.

**Procedure**

im-glcm image n-gl [#:dist 1] [#:p-max 255]

Returns the GLCM for image.

The procedure scales the original image (it brings its values in the range [0 (~ n-gl 1)]), then computes and returns the GLCM.

The returned GLCM is an ‘image’ composed four channels (four square matrices of size n-gl), corresponding to the Gray Level Co-occurrences, each calculated at a specific ‘angle’, respectively 0°, 45°, 90°, and 135°.

---

\(^{14}\) Since it is used as a factor in all three formulas, the final result obtained using log2 is equivalent to the result obtained using log multiplied by 1.4426950408889634

\(^{15}\) Guile-CV computes the difference average using all elements of the Px-y, by default, but offers this option as a courtesy, for users who would want to use Guile-CV as an immediate substitute to compute image texture measures for a (large) image set for which they would already have trained a classifier. It is not recommended to use it otherwise.
Features
The Guile-CV procedures and methods related to image features.

Procedures

**im-features** image l-image [#:n-label #f]

Returns a list of features, one list for each labeled object - including the background - in ascending order.

Notes: (a) *image* can either be an RGB or a GRAY image; (b) *l-image* is the 'corresponding' labeled image; (c) when used, the #:n-label optional keyword argument must be total number of label values used in l-image, as returned by [im-label], page 39, and [im-label-all], page 39.

The GRAY feature list values are:

- **area** The labeled object area in pixel
- **left top right bottom** The coordinates of the 'bounding box' labeled object
- **mean-x mean-y** Also sometimes called the 'centroid', these are the average of the x and y coordinates of all of the pixels in the labeled object. These two coordinate values are floating points, representing the 'mathematical position' of the mean x and y values of the labeled object
- **min max mean std-dev** The minimum, maximum, mean and standard gray deviation labeled object values
- **major-ev-x** major-ev-y minor-ev-x minor-ev-y Respectively the major and minor eigen vectors (https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors) x and y normalized coordinates: (= (sqrt (+ (expt x 2) (expt y 2))) 1)
- **major-axis minor-axis** Respectively the major and minor eigen values (https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors), optimized so that they actually correspond to major and minor radius of the ellipse covering the same area as the particle itself
- **angle** The angle of the major eigen vector axis, in degrees in the trigonometric circle reference system

---

16 Note that when passed to [im-crop], page 33, right and bottom must be increased by 1: (im-crop image left top (+ right 1) (+ bottom 1)).
17 Note that Vigra calculates and returns these values in the image coordinate system, where the y-axis is 'flipped' compared to the trigonometric circle 'traditional' representation. Guile-CV however transforms and returns these values using the trigonometric circle reference system.
center-mass-x center-mass-y
The center of mass x and y coordinates

perimeter
The labeled object perimeter in pixels

skewness kurtosis
Respectively the skewness (https://en.wikipedia.org/wiki/Skewness) and the kurtosis (https://en.wikipedia.org/wiki/Kurtosis) of the labeled object

circularity aspect-ratio roundness
Respectively the circularity (/ (* 4 %pi area) (expt perimeter 2)), the aspect ratio (/ major-axis minor-axis) and the roundness (/ minor-axis major-axis) of the labeled object

The RGB feature list values are:

area
The labeled object area in pixel

left top right bottom
The coordinates of the labeled object (the corresponding GRAY feature footnote applies here too of course)

mean-x mean-y
Also sometimes called the ‘centroid’, these are the average of the x and y coordinates of all of the (red green blue) pixels in the labeled object. These two coordinate values are floating points, representing the ‘mathematical position’ of the mean x and y values of the labeled object

min-r min-g min-b max-r max-g max-b mean-r mean-g mean-b std-dev-r std-dev-g std-dev-b
The minimum, maximum, mean and standard deviation labeled object values of the red, green and blue channels

major-axis minor-axis
Respectively the major and minor eigen values (https://en.wikipedia.org/wiki/Eigenvalues_and_eigenvectors), optimized so that they actually correspond to major and minor radius of the ellipse covering the same area as the particle itself

angle
The angle of the major eigen vector axis, in degrees in the trigonometric circle reference system

center-mass-x center-mass-y
The center of mass x and y coordinates

perimeter
The labeled object perimeter in pixels

skewness-r skewness-g skewness-b kurtosis-r kurtosis-g kurtosis-b
Respectively the skewness (https://en.wikipedia.org/wiki/Skewness) and the kurtosis (https://en.wikipedia.org/wiki/Kurtosis) labeled object values of the red, green and blue channels
circularity aspect-ratio roundness
Respectively the circularity \((\frac{4 \pi \text{area}}{\text{expt \ perimeter}^2})\), the aspect ratio \((\frac{\text{major-axis}}{\text{minor-axis}})\) and the roundness \((\frac{\text{minor-axis}}{\text{major-axis}})\) of the labeled object.

Though we did not make it public, Guile-CV has an internal feature display procedure that you might be interested to (re)use, so here is an example of a GRAY feature list display:

```scheme
scheme@(guile-user)> ,use (cv)
scheme@(guile-user)> (im-load "pp-17-bf.png")
$2 = (85 95 3 (f32(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... ...) ...))
scheme@(guile-user)> (im-rgb->gray $2)
$3 = (85 95 1 (f32(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 # ...)))
$4 = (0.0 251.0 128.3132714138286 8075)
scheme@(guile-user)> (im-threshold $3 136)
$5 = (85 95 1 (f32(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 # ...)))
scheme@(guile-user)> (im-label $5)
$6 = (85 95 1 (f32(0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 # ...)))
$7 = 2
scheme@(guile-user)> (im-features $2 $6)
$8 = ((3782 0 0 84 94 43.196319580078 45.67588958740234 0.0 # ...) ...)
scheme@(guile-user)> ((@@ (cv features) f-display) (match $8 ((bg a) a)))

area : 4293 (pixels)
left top right bottom : 0 0 84 94
mean-x mean-y : 40.94992 48.18262
min (red, green, blue) : 137.00000 136.00000 135.00000
max (red, green, blue) : 255.00000 250.00000 250.00000
mean (red, green, blue) : 236.13417 232.84999 232.84207
std. dev. (red, green, blue) : 20.23275 19.41402 19.84854
major ev x, y : 0.22202 0.97504
minor ev x, y : 0.97504 -0.22202
major, minor axis : 39.86419 34.27900 (radius)
gle : 77.17241 (degrees)
center of mass x, y : 40.73749 48.28692
perimeter : 367.74725
skewness (red, green, blue) : -2.90164 -2.99066 -2.91777
kurtosis (red, green, blue) : 8.53371 9.05482 8.61162
circularity : 0.39891
aspect ratio : 1.16293
roundness : 0.85989
```

As we mentioned above, f-display is defined in the (cv features) module, but it is not exported: in Guile, calling none exported procedure (which should not be ‘abused’) is done using the syntax @@ module-name binding-name, which in this example translates to @@ (cv features) f-display).
Particles
The Guile-CV procedures and methods to obtain and clean image particles.

Procedures

\texttt{im-particles image features [#:clean #t]} \hspace{1cm} \text{[Procedure]}

Returns two values, a list of images (the particles) and a list of their bounding boxes in the original \texttt{image}.

Each returned image is a ‘\texttt{particle}’, which is a subpart of \texttt{image} determined by its bounding box, given by the \texttt{left top right bottom} values of the corresponding ‘\texttt{entry}’ in \texttt{features} (see [im-features], page 23, for a complete description of a feature value list.

When \texttt{#:clean} is \texttt{#t}, which is the default, [im-particle-clean], page 26, is called upon each particle (see below for a description of the expected result).

\texttt{im-particle-clean particle} \hspace{1cm} \text{[Procedure]}

Returns a new image.

Cleaning a \texttt{particle} (which is an image) means detecting and removing any object(s) that is(are) not connected to the ‘\texttt{particle}’ itself.

This procedure is based on the property that in a ‘\texttt{particle}’, which is an (sub)image resulting from a [im-crop], page 33, based on the bounding box coordinates as returned by \texttt{im-features}, there precisely is one object that, if you call \texttt{im-features} upon \texttt{particle}, would have its bounding box coordinates being the entire \texttt{particle}. In other words, if you call \texttt{im-particle-clean} upon an image that is not a ‘\texttt{particle}’, the result will just be a black image.

Filter
The Guile-CV procedures and methods to filter images.

Procedures

\texttt{im-gaussian-blur image sigma} \hspace{1cm} \text{[Procedure]}

\texttt{im-gaussian-blur-channel channel width height sigma} \hspace{1cm} \text{[Procedure]}

Returns a new image or channel.

The new image or new channel is the result of the computation of the Gaussian blurring, also known as the Gaussian smoothing, by means of a convolution of \texttt{image} or \texttt{channel} with a 2D Gaussian function, where \texttt{sigma} is the standard deviation of the Gaussian distribution.

\texttt{im-gaussian-gradient image sigma} \hspace{1cm} \text{[Procedure]}

\texttt{im-gaussian-gradient-channel channel width height sigma} \hspace{1cm} \text{[Procedure]}

Returns a new image or channel.

The new image or new channel is the result of the computation of the strength of the first order partial derivatives by means of a convolution of \texttt{image} or \texttt{channel} with the first order derivative of a 2D Gaussian function, where \texttt{sigma} is the standard deviation of the Gaussian distribution.
\textbf{im-gaussian-sharp} \textit{image factor scale} \hfill \text{[Procedure]}

\textbf{im-gaussian-sharp-channel} \textit{channel width height factor scale} \hfill \text{[Procedure]}

Returns a new image or channel.

The new image or new channel is the result of the computation of the Gaussian sharpening: the procedure does (a) perform a Gaussian smoothing at the given scale to create a temporary image \textit{smooth} and (b) blends \textit{image} and \textit{smooth} according to the formula \((- (* (+ factor 1) image) (* smooth factor))).

\textbf{im-sharpen} \textit{image factor} \hfill \text{[Procedure]}

\textbf{im-sharpen-channel} \textit{channel width height factor} \hfill \text{[Procedure]}

Returns a new image or channel.

This procedure performs a \textit{simple sharpening} operation on \textit{image}. It actually calls \textbf{[im-convolve]}, page 27, with the following kernel:

\begin{align*}
\begin{bmatrix}
-1/16 & -1/8 & -1/16 & 0 & 0 & 0 \\
-1/8 & 3/4 & -1/8 & + & 0 & 1 & 0 \\
-1/16 & -1/8 & -1/16 & 0 & 0 & 0 \\
\end{bmatrix}
\end{align*}

and uses \textit{mirror} as the \textit{out of bound strategy}.

\textbf{im-median-filter} \textit{image w-width w-height} \hfill \text{[#:obs 'repeat]}

\textbf{im-median-filter-channel} \textit{channel width height w-width} \hfill \text{[#:obs 'repeat]}

Returns a new image or channel.

In the new image or channel, each pixel value is the \textit{median} value of neighboring entries. The pattern of neighbors is called a \textit{window}, the size of which is given by \textit{w-width} and \textit{w-height} \hspace{1em} (see Median Filter \hspace{1em} (https://en.wikipedia.org/wiki/Median_filter) for more information). Both \textit{w-width} and \textit{w-height} must be odd numbers, inferior to \textit{width} and \textit{height} respectively.

The optional keyword argument \textit{#:obs} determines the \textit{out-of-bound strategy}. Valid \textit{#:obs} symbols are:

\begin{itemize}
  \item \textbf{avoid} do not operate on pixels upon which (centering) the kernel does not fit in the image
  \item \textbf{repeat} repeat the nearest pixels
  \item \textbf{mirror} mirror the nearest pixels
  \item \textbf{wrap} wrap image around (periodic boundary conditions)
  \item \textbf{zero} out-of-bound pixel values to be 0.0
\end{itemize}

\textbf{im-convolve} \textit{image kernel} \hfill \text{[#:obs 'repeat]}

\textbf{im-convolve-channel} \textit{channel width height kernel k-width k-height} \hfill \text{[#:obs 'repeat]}

Returns a new image or channel.

The new image or new channel is the result of the convolution \hspace{1em} (https://en.wikipedia.org/wiki/Kernel_(image_processing)#Convolution)
k-width and k-height values can be different, but they must be odd numbers, inferior
to width and height respectively.

The optional keyword argument #:obs determines the ‘out-of-bound strategy’. Valid #:obs symbols are:

- avoid: do not operate on pixels upon which (centering) the kernel does not fit in the image
- clip: clip the kernel when operating on pixels upon which (centering) the kernel does not fit in the image (this is only useful if the kernel is >= 0 everywhere)
- repeat: repeat the nearest pixels
- mirror: mirror the nearest pixels
- wrap: wrap image around (periodic boundary conditions)
- zero: out-of-bound pixel values to be 0.0

Kernel data structure, accessors, procedures and predefined kernels are all described

im-nl-means image arg... [Procedure]
im-nl-means-channel channel width height arg... [Procedure]

Returns a new image or channel.

The new image or new channel is the result of a non-local means
(https://en.wikipedia.org/wiki/Non-local_means) denoising as described here\textsuperscript{18}. The following table lists the optional keyword arguments and their default values:

Policy arguments:

```
#:policy-type
accepts 0 (ratio policy) or 1 (norm policy)
#:sigma 15.0
default to 5.0 if the policy-type is 0
#:mean-ratio 5.0
default to 0.95 if the policy-type is 0
#:variance-ratio 0.5
#:epsilon 1.0e-5
```

Filter arguments:

```
#:spatial-sigma 2.0
#:search-radius 3
#:patch-radius 1
```

the patch-radius can be either 1 or 2

The `im-nl-means-channel` procedure accepts one additional optional keyword argument:

```
#:n-thread (- (current-processor-count) 1)
```

FIXME need to describe the parameters

**Process**

The Guile-CV procedures and methods to process images.

**Procedures**

**im-threshold** *image threshold [#:bg 'black] [Procedure]

Returns a new BLACK and WHITE image.

The *image* argument can either be a GRAY or an RGB image, in which case each pixel is converted to GRAY as it is processed. Valid #:bg values are black (the default) and white.

Pixels for which the original value is >= *threshold* are set to 255.0 if #:bg is 'black, and set to 0.0 if #:bg is 'white. The other pixels are set to 0.0 or 255.0 respectively.

**im-scrap** *image size [#:pred <] [#:con 8] [#:bg 'black] [#:exclude-on-edges #f] [Procedure]

Returns a new image.

Scraping an image is the operation of removing objects depending on their size (in pixels). When exclude-on-edges is #t, all objects that are on any edges are also removed.

The procedure first calls [im-label], page 39, using con and bg, then calls [im-features], page 23. The area feature of each object is then compared to size using pred and the object is removed if the result is #t.

Note that image must be a binary image.

**im-add image val** [Method]

**im-add i1 i2 i3 . . .** [Method]

**im-add-channel channel width height val** [Method]

**im-add-channel width height c1 c2 c3 . . .** [Method]

Returns a new image or channel.

Performs the scalar addition of image with val or the matrix addition of i1 i2 i3 . . . or c1 c2 c3 . . . respectively.

**im-subtract image val** [Method]

**im-subtract i1 i2 i3 . . .** [Method]

**im-subtract-channel channel width height val** [Method]

**im-subtract-channel width height c1 c2 c3 . . .** [Method]

Returns a new image or channel.
Performs the scalar subtraction of image with val or the matrix subtraction of \(i1\) \(i2\) 
\(i3\) ... or \(c1\) \(c2\) \(c3\) ... respectively.

\[
\text{im-times} \quad \text{image val} \\
\text{im-times} \quad i1 \ i2 \ i3 \ldots \\
\text{im-times-channel} \quad \text{channel width height val} \\
\text{im-times-channel} \quad c1 \ w1 \ h1 \ c2 \ w2 \ h2 \ c3 \ w3 \ h3 \ldots
\]

Returns a new image or channel.

Performs the scalar multiplication of image with val or the element by element multiplication of \(i1\) \(i2\) \(i3\) ... or \(c1\) \(c2\) \(c3\) ... respectively.

\[
\text{im-mtimes} \quad i1 \ i2 \ i3 \ldots \\
\text{im-mtimes-channel} \quad \text{width height} \ c1 \ c2 \ c3 \ldots
\]

Returns a new image or channel.

Performs matrix multiplication of \(i1\) \(i2\) \(i3\) ... or \(c1\) \(w1\) \(h1\) \(c2\) \(w2\) \(h2\) \(c3\) \(w3\) \(h3\) ... recursively. The number of lines of the next image must equal the number of columns of the previous intermediate result.

\[
\text{im-divide} \quad \text{image val} \\
\text{im-divide} \quad i1 \ i2 \ i3 \ldots \\
\text{im-divide-channel} \quad \text{channel width height val} \\
\text{im-divide-channel} \quad c1 \ w1 \ h1 \ c2 \ w2 \ h2 \ c3 \ w3 \ h3 \ldots
\]

Returns a new image or channel.

Performs the scalar division of image with val or the element by element division of \(i1\) \(i2\) \(i3\) ... or \(c1\) \(c2\) \(c3\) ... respectively.

It is the user responsibility to insure that none of the \(c2\) \(c3\) ... values is zero.

\[
\text{im-mdivide} \quad i1 \ i2 \ i3 \ldots \\
\text{im-mdivide-channel} \quad \text{width height} \ c1 \ c2 \ c3 \ldots
\]

Returns a new image or channel.

Performs the matrix multiplication of \(i1\) or \(c1\) by the inverse of \(i2\) \(i3\) ... or \(c2\) \(c3\) ... recursively. The number of lines of the next image must equal the number of columns of the previous intermediate result\(^{19}\).

It is the user responsibility to insure that none of the \(c2\) \(c3\) ... values is zero.

\[
\text{im-range} \quad \text{image} \\
\text{im-range-channel} \quad \text{channel width}
\]

Returns a list of six values (min row col max row col) if image is GRAY, and a list of list of these values if image is RGB or for any n-chan > 1 images.

\[
\text{im-min} \quad \text{image} \\
\text{im-max} \quad \text{image} \\
\text{im-min-channel} \quad \text{channel width}
\]

\(^{19}\) Technically speaking, there is no such thing as matrix division. Dividing a matrix by another matrix is an undefined function. The closest equivalent is to multiply the matrix by the inverse of the other matrix.
\textbf{im-max-channel} \textit{channel width}

Returns three multiple values if \textit{image} is GRAY: \textit{min row col} or \textit{max row col} respectively. If \textit{image} is RGB or for any \textit{n-chan} > 1 images, it returns a list of list of these values.

\textbf{im-map} \textit{proc i1 i2 i3 \ldots}

\textbf{im-map-channel} \textit{proc width height c1 c2 c3 \ldots}

Returns a new image or channel.

Apply \textit{proc} to each pixel value of each channel of \textit{i1} (if only two arguments are given), or to the corresponding pixel values of each channels of \textit{i1 i2 i3 \ldots} (if more than two arguments are given).

\textbf{im-reduce} \textit{image proc default}

\textbf{im-reduce-channel} \textit{channel proc default}

Returns one value if \textit{image} is GRAY. If \textit{image} is RGB or for any \textit{n-chan} > 1, it returns a list of values.

If \textit{image} is empty, \textbf{im-reduce} returns \textit{default} (this is the only use for default). If \textit{image} has only one pixel, then the pixel value is what is returned. Otherwise, \textit{proc} is called on the pixel values of \textit{image}.

Each \textit{proc} call is \textit{(proc elem prev)}, where \textit{elem} is a pixel value from the channel (the second and subsequent pixel values of the channel), and \textit{prev} is the returned value from the previous call to \textit{proc}. The first pixel value - for each channel - is the \textit{prev} for the first call to \textit{proc}.

For example:

```
(use (cv))
(im-load "edx.png")
\(-\)
$2 = (128 128 1 (#f32(4.0 26.0 102.0 97.0 58.0 10.0 9.0 21.0 # ...)))
(im-reduce $2 + 0)
\(-\)
$3 = 556197.0
```

\textbf{im-normalize} \textit{image} \textit{[#:value 255.0]}

\textbf{im-normalize-channel} \textit{channel width height} \textit{[#:value 255.0]}

Returns a new normalized image or channel.

Normalizing an \textit{image} or a \textit{channel} consist of dividing all pixels by a value so they all fall in the \([0.0 \rightarrow 1.0]\) range. The default \textit{#:value} is 255.0.

\textbf{im-and} \textit{i1 i2 i3 \ldots}

\textbf{im-and-channel} \textit{width height c1 c2 c3 \ldots}

\textbf{im-or} \textit{i1 i2 i3 \ldots}

\textbf{im-or-channel} \textit{width height c1 c2 c3 \ldots}

\textbf{im-xor} \textit{i1 i2 i3 \ldots}

\textbf{im-xor-channel} \textit{width height c1 c2 c3 \ldots}

Returns \textit{image} if one argument only, otherwise, it returns a new image or channel, as the result of computing the logical AND, OR or XOR of all images or channels.
In the case of **AND**, for all positive results, the pixel values (of each channel) of the new image are set to the one obtained from \( i1 \) or \( c1 \) respectively, and 0.0 otherwise.

In the case of **OR**, the pixel values (of each channel) of the new image are set to the one obtained from the first non zero \( ii \) or \( ci \) respectively, otherwise it is set to 0.0.

In the case of **XOR**, the pixel values (of each channel) of the new image are set to the value obtained from successively computing \( \text{logior} \ (\text{logand} \ a \ (-255 \ b)) \ (\text{logand} \ (-255 \ a) \ b) \) where \( a \) would be the previous result and \( b \) the current image or channel pixel value, until all images passed in arguments have been processed\(^{20}\).

All images must have the same width, height and n-channel.

There are, of course, scientific use and examples of images logical XOR, and that is why Guile-CV (http://www.gnu.org/software/guile-cv) is being developed for, but let’s have a bit of fun here, and see if our levitating GNU likes apples!

**Transform**

The Guile-CV procedures and methods to transform images.

**Procedures**

\[
\begin{align*}
im-rgba->rgb & \quad \text{image} \ [\#:bg \ '(0.0 0.0 0.0)] \quad \text{[Procedure]} \\
im-rgba->gray & \quad \text{image} \ [\#:bg \ '(0.0 0.0 0.0)] \quad \text{[Procedure]} \\
im-rgb->gray & \quad \text{image} \quad \text{[Procedure]} \\
& \quad \text{Returns a new RGB or GRAY image.}
\end{align*}
\]

In the RGBA case, image channels are first normalized. The new RGB channels are obtained by applying the following pseudo code algorithm:

\[
\begin{align*}
R &= (((1 - \text{Source}.A) \times \text{BG}.R) + (\text{Source}.A \times \text{Source}.R) \times 255.0 \\
G &= (((1 - \text{Source}.A) \times \text{BG}.G) + (\text{Source}.A \times \text{Source}.G) \times 255.0 \\
B &= (((1 - \text{Source}.A) \times \text{BG}.B) + (\text{Source}.A \times \text{Source}.B) \times 255.0 \\
im-resize & \quad \text{image} \ \text{new-width} \ \text{new-height} \ [\#:i-mode \ 'bilinear] \quad \text{[Procedure]} \\
im-resize-channel & \quad \text{channel} \ \text{width} \ \text{height} \ \text{new-width} \ \text{new-height} \ [\#:i-mode \ 'bilinear] \\
& \quad \text{[Procedure]} \\
& \quad \text{Returns a new image or channel resized to new-width, new-height.}
\end{align*}
\]

The interpolation mode \#:i-mode, can be one of:

- none
- bilinear
- biquadratic
- bicubic
- ? (fixme)

\[
\begin{align*}
im-rotate & \quad \text{image} \ \text{angle} \ [\#:i-mode \ 'bilinear] \\
im-rotate-channel & \quad \text{channel} \ \text{width} \ \text{height} \ \text{angle} \ [\#:i-mode \ 'bilinear] \\
& \quad \text{[Procedure]} \\
& \quad \text{Returns a new image or channel rotated by angle.}
\end{align*}
\]

\(^{20}\) Note that there is no mathematically valid XOR operation on floating points, hence as they are ‘accessed’, pixel values are converted to integer, using \texttt{float->int}, defined in the (cv support libguile-cv) module.
The *angle* is in degrees: \(+/-(0.0 \text{ to } 360.0)\).

It is necessary, for rotations other than multiples of 90°, to recalculate the target coordinates, since after the rotation, they might be floats. The ‘next neighbor’ interpolation possible modes, `#:i-mode`, are:

- `bilinear`
- `biquadratic`
- `bicubic`
- `? (fixme)`

**im-flip** *image plane*  
**im-flip-channel** *channel width height plane*  
Returns a new image or channel flipped according to the selected *plane*. Valid flipping *plane* values are:

- `hori horizontal`
- `vert vertical`
- `both`

**im-invert** *image*  
**im-invert-channel** *channel width height*  
Returns a new inversed image or channel. Calculating the inverse of an *image* or a *channel* consist of applying the exponent function, `expt`, to all pixel values, raising them to the power of -1.

**im-transpose** *image*  
**im-transpose-channel** *channel width height*  
Returns a new tranposed image or channel. Transposing an *image* or a *channel* consist of flipping it over its main diagonal. In the transposed result, switched in size, row values are the original column values and column values are the original row values.

**im-complement** *image*  
Returns a new image. This procedure computes the mathematical complement of *image*, which for Guile-CV means that for each pixel of each channel, the new value is \((-255.0 \text{ pixel-value})\).

**im-clip** *image* [#lower 0.0] [#upper 255.0]  
**im-clip-channel** *channel width height* [#lower 0.0] [#upper 255.0]  
Returns a new clipped image or channel. Clipping an *image* or a *channel* consist of replacing all pixel values below *lower* by the *lower* value and all pixel values above *upper* by the *upper* value.

**im-crop** *image left top right bottom*  
**im-crop-channel** *channel width height left top right bottom*  
[#:new-w #f] [#:new-h #f]  
Returns a new image, resulting of the crop of *image* at *left*, *top*, *right* and *bottom*. 
**Procedure**
im-crop-size width height left top right bottom

Returns a list, \((\text{new-width new-height})\).

Given the original image width and height, this procedure checks that left, top, right and bottom are valid and return a list, \((\text{new-width new-height})\), otherwise, it raises an error.

**Procedure**
im-padd image left top right bottom [#color (0.0 0.0 0.0)]
im-padd-channel channel width height left top right bottom [#:new-w #f] [#:new-h #f] [#:value 0.0]

Returns a new image or channel, respectively padding image or channel by left, top, right and bottom pixels initialized respectively to color or value. Note that when im-padd is called upon a GRAY image, color is reduced to its corresponding gray value:

\[
(\text{reduce } + 0 \text{ color} 3)
\]

**Procedure**
im-padd-size width height left top right bottom

Returns a list, \((\text{new-width new-height})\).

Given the original image width and height, this procedure checks that left, top, right and bottom are >= 0 and return a list, \((\text{new-width new-height})\), otherwise, it raises an error.

**Procedure**
im-local-minima image [#:threshold +float-max+]
im-local-maxima image [#:threshold (-+float-max+)]
im-local-minima-channel channel width height [#:threshold +float-max+]
im-local-maxima-channel channel width height [#:threshold (-+float-max+)]

All local minima and maxima related procedures also accept the following additional optional keyword arguments: [#:con 8] [#:marker 1.0] [#:borders? #f] [#:plateaus? #f] [#:epsilon 1.0e-4]

Returns a new image or channel.

Finds the local minima or maxima in image or channel. Local minima or maxima are defined as ‘points’ that are not on the borders (unless #:borders? is #t), and whose values are lower or higher, respectively, then the values of all direct neighbors. In the result image or channel, these points are marked using the #:marker value (all other pixels values will be set to 0).

By default, the algorithm uses 8-connectivity to define a neighborhood, which can be changed passing the optional keyword argument #:con, which can be either 4 or 8.

The #:threshold optional keyword argument can be used to discard minima and maxima whose (original pixel) value is not below or above the threshold, respectively. Both default values depend on +float-max+, which is defined (and so is +float-min+) using the corresponding limit value as given by the C float.h header.

The #:plateaus? optional keyword argument can be used to allow regions of ‘constant’ (original pixel) value whose neighbors are all higher (minima) or lower (maxima) than the value of the region. Tow pixel values are considered part of the
same region (representing the same ‘\texttt{constant}’ value) if the absolute value of their
difference is $\leq$ to $\#\texttt{epsilon}$.

Notes:
- If you want to know how many minima or maxima were found, use [im-reduce],
  page 31, upon the result;
- If you are interested by the original minima or maxima pixel values, Use [im-
times], page 30, between the original image and the result.

\begin{verbatim}
im-fft image
im-fft-channel channel width height
\end{verbatim}

Returns two images or channels.
Computes and returns the Fast Fourier Transform\footnote{The FFT (Fast Fourier Transform) is simply a faster way to compute the Fourier transform. All FFT related procedures, and their inverse, are obtained by calling the FFTW library (\url{http://www.fftw.org/}).} of the \texttt{image} or \texttt{channel}. It returns two values, images or channels: the first contains the real part and the second the imaginary part of the transformation.

\begin{verbatim}
im-fft-inverse real imaginary
im-fft-inverse-channel real-channel imaginary-channel width height
\end{verbatim}

Returns two images or channels.
Computes and returns the inverse Fast Fourier Transform given its \texttt{real} and \texttt{imaginary} or \texttt{real-channel} and \texttt{imaginary-channel} parts. It returns two values, images or channels: the first contains the real part and the second the imaginary part of the inverse transformation.

\begin{verbatim}
im-fcc image mask
im-fncc image mask
im-fcc-channel i-chan m-chan width height m-width m-height
im-fncc-channel i-chan m-chan width height m-width m-height
\end{verbatim}

Returns an image or a channel.
Computes and returns the Fast Cross Correlation or Fast Normalized Cross Correlation between \texttt{image} and a (usually smaller) \texttt{mask}\footnote{Also called a \texttt{template}, or a \texttt{pattern}}\footnote{Hence the names, FCC (Fast Cross Correlation) and FNCC (Fast Normalized Cross Correlation).}, using the Fast Fourier Transform\footnote{Hence the names, FCC (Fast Cross Correlation) and FNCC (Fast Normalized Cross Correlation).}.

\textbf{Morphology}

The Guile-CV procedures and methods related to morphology.

\textbf{Procedures}

\begin{verbatim}
im-disc-erode image radius
im-disc-erode-channel channel width height radius
\end{verbatim}

Returns a new image or channel.
Performs the morphological erosion of \textit{image} using a disc of a given \textit{radius}. Here is an example:

\begin{verbatim}
(im-make 5 5 1 1.0)
| $2 = (5 5 1 (#f32(1.0 1.0 1.0 1.0 1.0 ...)))
(im-set! $2 1 2 0.0)
(im-disc-erode $2 1)
| $3 = (5 5 1 (#f32(1.0 0.0 0.0 0.0 1.0 ...)))
(im-display $2 #:proc inexact->exact)
| Channel 1
1 1 1 1 1
1 1 0 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
(im-display $3 #:proc inexact->exact)
| Channel 1
0 1 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 1 1 1 1
\end{verbatim}

\textbf{im-disc-erode \textit{image} \textit{radius}} \hfill [Procedure]
\textbf{im-disc-dilate-channel \textit{channel} \textit{width} \textit{height} \textit{radius}} \hfill [Procedure]

Returns a new image or channel.

Performs the morphological dilation of \textit{image} using a disc of a given \textit{radius}. Here is an example:

\begin{verbatim}
...  
| $13 = (11 11 1 (#f32(0.0 0.0 0.0 0.0 0.0 ...)))
(im-disc-dilate $13 1)
| $14 = (11 11 1 (#f32(1.0 1.0 1.0 1.0 1.0 ...)))
(im-display $13 #:proc inexact->exact)
| Channel 1
0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 1 1 1 0
0 1 1 1 1 0 0 1 1 1 0
0 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 0
0 1 1 0 0 0 1 1 1 1 0
\end{verbatim}
im-open image radius  

| Procedure |

Implements a new image or channel.

Performs the dilation of the erosion of image using radius. Opening removes small objects.

im-open-channel channel width height radius  

| Procedure |

Implements a new image or channel.

Performs the dilation of the erosion of image using radius. Closing removes small holes.

im-close image radius  

| Procedure |

Implements a new image or channel.

Performs the erosion of the dilation of image using radius. Closing removes small holes.

im-close-channel channel width height radius  

| Procedure |

Implements a new image or channel.

Performs the erosion of the dilation of image using radius. Closing removes small holes.

im-fill-holes image  

| Procedure |

Implements a new image or channel.

The argument must be a BINARY image. As its name indicate, this procedure fill the holes of all and every objects in the image.

im-fill-holes-channel channel width height  

| Procedure |

Implements a new image or channel.

The argument must be a BINARY image. As its name indicate, this procedure fill the holes of all and every objects in the image.

im-delineate image [#:threshold 10] [#:radius 2]  

| Procedure |

Implements a new image or channel.

Both threshold and radius must be exact integers.

Also known as ‘Edge Enhancement’, this procedure performs the delineation of image, obtained by applying the following pseudo code algorithm:

```;;; with```
III. Guile-CV Core Reference

```lisp
;; Min = (im-disc-erode image radius)
;; Max = (im-disc-dilate image radius)
D = Max - Min
If D < threshold
    ;; not an edge
    output pixel = input pixel
    ;; it is an edge
    If (pixel -- Min) < (Max -- pixel)
        output pixel = Min
        output pixel = Max
```

Here above, left being the original image - a small part of an optical microscope capture of a sinter sample - you can see the difference between `im-delineate` called with the default `threshold` and `radius` values, then called using `#:threshold 25` and `#:radius 5`.

`im-distance-map image [#:bg 'black'] [#:mode 'euclidean]` [Procedure]

`im-distance-map-channel channel width height [#:bg 'black'] [#:mode 'euclidean]` [Procedure]

Returns a new image or channel.

Also know as ‘Distance Tranform’, this procedure performs the distance map of `image`, which consist of, for each background pixel, calculating its distance to the nearest object or contour. In the return new image or channel, all background pixels will be assigned the their distance value, all other pixels will be assigned to 0. Distances larger than 255 are labelled as 255.

The default background pixel value is `'black`, the optional `#:bg` keyword argument also accepts `'white`.


Here above, left being the original image - a few cells - you can see the results obtained by calling `im-distance-map` using respectively the `'euclidean`, `'manhattan` and `'chessboard` modes.

`im-reconstruct image seeds [#:con 8]` [Procedure]

Returns a new image.

This procedure implements a ‘binary morphological reconstruction’ algorithm, which extracts the connected components of `image` that are ‘marked’ by (any of) the connected components contained in `seeds`.

Morphological reconstruction is part of a set of image operators often referred to as ‘geodesic’ (geodesic distance, geodesic dilation ...). Morphological (or geodesic) operations upon digital images come from and use the Mathematical morphology (MM) (https://en.wikipedia.org/wiki/Mathematical_morphology) theory and technique developed for the analysis and processing of geometrical structures.

### Segmentation

The Guile-CV procedures and methods related to segmentation.

#### Procedures

- **im-label** `image` [#con 8] [#bg 'black]

- **im-label-channel** `channel` `width` `height` [#con 8] [#bg 'black]

- **im-label-all** `image` [#con 8]

- **im-label-all-channel** `channel` `width` `height` [#con 8]

  Returns two values: a new GRAY image or channel, and the total number of labels\textsuperscript{25}.

The **im-label** and **im-label-channel** procedures label foreground objects in the binary `image`. In the new image or channel, 0.0 indicates a background pixel, 1.0 indicates that the pixel belongs to object number 1, 2.0 that the pixel belongs to object number 2, etc.

The **im-label-all** and **im-label-all-channel** procedures label all objects in the binary `image`, with no specific distinction for any background value. As a result, these two procedures will label not only the continuous background, if any, but also any hole(s). As an example, they are used by **[im-fill-holes]**, page 37, defined in the module (**cv morphology**), which you may have a look at for a better understanding of how it works.

Two pixels belong to the same object if they are neighbors. By default the algorithm uses 8-connectivity to define a neighborhood, but this can be changed through the keyword argument `#:con`, which can be either 4 or 8.

- **im-canny** `image` [#sigma 1.0] [#threshold 0.0] [#marker 255.0]

- **im-canny-channel** `channel` `width` `height` [#sigma 1.0] [#threshold 0.0] [#marker 255.0]

  Returns a new image or channel.

Detect and mark edges using a Canny Edge Detector (https://en.wikipedia.org/wiki/Canny_edge_detector) algorithm: (a) compute the `image` Gaussian gradient using `sigma`, (b) remove edges whose strength is below `threshold`, then for all remaining edges, (d) remove the non-local maxima (edge thinning (https://en.wikipedia.org/wiki/Edge_detection#Edge_thinning)) and (e) set their intensity using `marker`.


\textsuperscript{25} The number of labels correspond to the highest label value + 1: earlier version of Guile-CV, prior to version 1.8.0, did return the number of objects, which correspond to the highest label value. This was less then optimal, since not only 0.0 is a label, but other procedures, im-features for example, do consider and return and element for the background as well.
III. Guile-CV Core Reference

**Procedure**
im-crack-edge  image [#:marker 255.0]  [Procedure]
im-crack-edge-channel channel width height [#:marker 255.0]  [Procedure]

Returns a new image or channel.

Crack edges are marked ‘between’ the (different) pixels of image. In order to accommodate the cracks, the resulting image or channel size must be (- (* width 2) 1) and (- (* height 2) 1) respectively.

Crack pixels are first inserted, then all crack pixels whose non-crack neighbors have different values are crack edges and marked using marker, while all other pixels (crack and non-crack) become region pixels. Here is a simple example, with two regions, a and b, and using * as the crack edge marker:

<table>
<thead>
<tr>
<th>Original</th>
<th>Inserted Cracks</th>
<th>Final Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b b</td>
<td>. b . b</td>
<td>a * b b b</td>
</tr>
<tr>
<td>a a b</td>
<td>. . . .</td>
<td>a * * * b</td>
</tr>
<tr>
<td>a a a</td>
<td>a . a . b</td>
<td>a a a * b</td>
</tr>
<tr>
<td>. . .</td>
<td>a a a *</td>
<td>a a a a a</td>
</tr>
<tr>
<td>a . a . a</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Crack Edge Images have the following properties:
- Crack Edge Images have odd width and height.
- Crack pixels have at least one odd coordinate.
- Only crack pixels may be marked as crack edge pixels.
- Crack pixels with two odd coordinates must be marked as edge pixels whenever any of their neighboring crack pixels was marked.

As a consequence of the last two properties, both edges and regions are 4-connected. Thus, 4-connectivity and 8-connectivity yield identical connected components in Crack Edge Images (the so called well-composedness). This ensures that Crack Edge Images have nice topological properties.

**Compose**

Other Guile-CV procedures and methods to compose images.

**Procedures**
im-compose position alignment [#:color '(0 0 0)] img-1 img-2 . . .  [Procedure]
im-compose-channels position alignment channels widths heights [#:value '0.0]  [Procedure]

Returns a new image or a new channel.

The valid position and alignment symbols are:

left right
top center bottom

---

When used, the optional #:color keyword argument must come after the mandatory alignment argument and precede img-1, otherwise Guile will raise an exception. For RGB images, it is the color used to pad images passed in argument that are inferior, in width or height (depending on the position), to the biggest of them. For GRAY images, the #:color is reduced to its corresponding gray value:

\[
\text{(reduce + 0 color) 3}
\]

For the im-compose-channels procedure, the list of channels, widths and heights must be of equal length and equally ordered, so the nth element of widths and heights are the width and height of the nth element of channels. The optional #:value keyword argument is used to pad channels that are inferior, in width or height (depending on the position), to the biggest of them.

Utilities

Other Guile-CV utility procedures, methods and variables.

Procedures

Variables

Procedures

\text{im-display } \text{image} \ [\#:proc \#f] \ [\#:port (current-output-port)] \ [Procedure]

\text{im-display-channel } \text{channel width height} \ [\#:proc \#f] \ [\#:port (current-output-port)] \ [Procedure]

Returns nothing.

Displays the content of image or channel on port.

The optional #:proc keyword argument must either be #f, the default, or a procedure that accepts a single (32 bits float) argument. When #:proc is #f, im-display will use an internally defined procedure which formats its argument `à la octave': nine positions, six decimals, all number aligned on the dot. any value >= 1000 is converted to use the exponential float notation. Here is an 'hand made' example:

\[
\ldots
\]

\[
\text{scheme@(guile-user)> (im-divide$299)}
\]

\[
\text{scheme@(guile-user)> (im-set!$300010000)}
\]

\[
\text{scheme@(guile-user)> (im-display$4)}
\]

Channel 1

\[
1.0E+4 \quad 0.01010 \quad 0.02020 \quad 0.03030
\]
### III. Guile-CV Core Reference

| 0.04040 | 0.05051 | 0.06061 | 0.07071  |
| 0.08081 | 0.09091 | 0.10101 | 0.11111  |

**Channel 2**

| 0.12121 | 0.13131 | 0.14141 | 0.15152  |
| 0.16162 | 0.17172 | 0.18182 | 0.19192  |
| 0.20202 | 0.21212 | 0.22222 | 0.23232  |

**Channel 3**

| 0.24242 | 0.25253 | 0.26263 | 0.27273  |
| 0.28283 | 0.29293 | 0.30303 | 0.31313  |
| 0.32323 | 0.33333 | 0.34343 | 0.35354  |

**Caution:** unless you specify `port`, both this and `im-display-channel`, page 41, procedures are meant to be used on very small and testing images, otherwise even on a small image, it might be ok in a terminal, but it will definitely will kill your `emacs`.

#### `im-show filename`

- **[Method]**
- Returns the string "#<Image: ...>", where "..." is either `filename` or a filename constructed by `im-show`, see below.

The optional `scale` argument can take the following values:

- **#f**
  - pixel values are `clipped`: values < 0 are saved as 0, values > 255 are saved as 255, and otherwise are saved unchanged

- **#t**
  - all pixel values are scaled\(^2\) to the [0 255] range

These three methods will also effectively display the image if you are using Geiser (http://www.nongnu.org/geiser), which analyzes Guile’s procedures and methods returned values (through the use of its pattern matcher), and when appropriate, triggers its image display mechanism.

Geiser has two variables that allow you to choose either to inline images in its Emacs (https://www.gnu.org/software/emacs) (Guile repl) buffer, or to display them using external viewer: `geiser-image-viewer` and `geiser-repl-inline-images-p`.

You may choose to add these variables in your `.emacs` file, for example:

```
(setq geiser-image-viewer "eog")
(setq geiser-repl-inline-images-p nil)
```

Note that `(setq geiser-repl-inline-images-p t)` will only work if you are using a graphics-aware Emacs, and otherwise, will fall on the external viewer approach, if the variable `geiser-image-viewer` has been defined. When using Geiser in a non graphics-aware Emacs, or when using the external viewer approach, images will appear

---

\(^2\) Note that in this particular context, `scale` does not mean a change in dimension, but rather bringing pixel values from the range they occupy in memory to the [0 255] range.
as buttons: press return on them to invoke (or raise) the external viewer (window containing that image).

Except for the first `im-show` method, Guile-CV has to save the image first, and does it in the location defined by the [%image-cache], page 43, variable. If you call `im-show` passing name, the image is saved as %image-cache/name.png, otherwise under a generated name, the result of (symbol->string (gensym "im-show-"))

Note that if you do not specify name, a new external viewer window is opened at each `im-show` invocation, even for identical image calls: this because in Guile-CV, on purpose, images are just list, with no (unique) identifier, and there is no way for `im-show` to know ... Further to this point, when you pass name as an argument, you are not ‘identifying’ image, which may actually differ, but rather just ask to reuse the filename and hence the external viewer window associated with it.

Last note: many external viewers, such as Eog (the Gnome Eye Viewer), will try to apply, per default, some sort of smoothing techniques, especially on zoom-in and zoom-out: where this is fine for viewing ’lazer’ pictures, you probably want to check and disable these options when working with Guile-CV.

Variables

%image-cache [Variable]

Specifies the location used by [im-show], page 42, to save images.

The default value is /tmp/<username>/guile-cv, but you may set! it. If you’d like to reuse that location for future guile-cv sessions, you may save it in guile-cv’s ‘per user’ config file <userdir>/.config/guile-cv as an assoc pair, here is an example:

```
cat ~/.config/guile-cv.conf
((image-cache . "~/tmp"))
```

Note that if used, the ‘~’ is expanded at load time, so in geiser, it becomes:

```
scheme@(guile-user)> ,use (cv)
scheme@(guile-user)> %image-cache
-| $2 = "/home/david/tmp"
```

%image-cache-format [Variable]

Specifies the format used by [im-show], page 42, to save images.

The default value is "png", but you may set! it. If you’d like to reuse that format for future guile-cv sessions, you may save it in guile-cv’s ‘per user’ config file <userdir>/.config/guile-cv, as an assoc pair, here is an example:

```
cat ~/.config/guile-cv.conf
((image-cache-format . "jpg"))
```

Support

Guile-CV uses a series of support modules, each documented in the following subsections. You may either import them all, like this (use-modules (cv support)), or individually, such as (use-modules (cv support modules)), (use-modules (cv support goops)), ...
Modules

[re-export-public-interface], page 44

re-export-public-interface mod1 mod2 ...

Syntax

Re-export the public interface of a mod1 mod2 ...

Invoked like use-modules, where each mod1 mod2 ... is a module name (a list of symbol(s)).

Goops

Pi

Procedures

radian->degree rad

Procedure

degree->radian deg

Procedure

Returns respectively a degree or a radian value.

Variables

%pi

Variable

%2pi

Variable

%pi/2

Variable

Respectively bound to (acos -1), (* 2 %pi) and (/ %pi 2).

Utils

Appendix A GNU Free Documentation License

http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document free in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or non-commercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals
providing the same freedoms that the software does. But this License is not limited to
software manuals; it can be used for any textual work, regardless of subject matter or
whether it is published as a printed book. We recommend this License principally for
works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a
notice placed by the copyright holder saying it can be distributed under the terms
of this License. Such a notice grants a world-wide, royalty-free license, unlimited in
duration, to use that work under the conditions stated herein. The “Document”,
below, refers to any such manual or work. Any member of the public is a licensee, and
is addressed as “you”. You accept the license if you copy, modify or distribute the work
in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or
a portion of it, either copied verbatim, or with modifications and/or translated into
another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document
that deals exclusively with the relationship of the publishers or authors of the Document
to the Document’s overall subject (or to related matters) and contains nothing that
could fall directly within that overall subject. (Thus, if the Document is in part a
textbook of mathematics, a Secondary Section may not explain any mathematics.) The
relationship could be a matter of historical connection with the subject or with related
matters, or of legal, commercial, philosophical, ethical or political position regarding
them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as
being those of Invariant Sections, in the notice that says that the Document is released
under this License. If a section does not fit the above definition of Secondary then it is
not allowed to be designated as Invariant. The Document may contain zero Invariant
Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover
Texts or Back-Cover Texts, in the notice that says that the Document is released under
this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may
be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented
in a format whose specification is available to the general public, that is suitable for
revising the document straightforwardly with generic text editors or (for images com-
posed of pixels) generic paint programs or (for drawings) some widely available drawing
editor, and that is suitable for input to text formatters or for automatic translation to
a variety of formats suitable for input to text formatters. A copy made in an otherwise
Transparent file format whose markup, or absence of markup, has been arranged to
thwart or discourage subsequent modification by readers is not Transparent. An image
format is not Transparent if used for any substantial amount of text. A copy that is
not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without
markup, Texinfo input format, LaTPeX input format, SGML or XML using a publicly
available DTD, and standard-conforming simple HTML, PostScript or PDF designed
for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document.
as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.
The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.
ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts being list.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.
Concept Index

This index contains concepts, keywords and non-Schemey names for several features, to make it easier to locate the desired sections.

C
Copying ........................................ 1
Crack Edge Image .......................... 40

D
Distance Map ................................. 38
Distance Transform ......................... 38

E
Edge Detection ................................... 39
Edge Enhancement .......................... 37
Euclidean Distance ......................... 38

F
Fast Cross Correlation ....................... 35
Fast Fourier Transform ..................... 35
Fast Fourier Transform Inverse .......... 35
Fast Normalized Cross Correlation .... 35
Features ....................................... 23

G
Gaussian Blur ..................................... 26
Gaussian Gradient ........................... 26
Gaussian Sharp ............................... 27
Gaussian Smooth ............................. 26
Geodesic Operators ......................... 38
GLCM ......................................... 22
GLCP ......................................... 22
GPL ............................................. 1
Gray Level Co-occurence Matrix ......... 22

I
Image Convolution ............................ 27
Image Particles .............................. 26
Image Textures ............................... 21

L
License ........................................... 1

M
Mathematical Morphology .................. 38
Matrix Division .............................. 30
Matrix Multiplication ...................... 30
Median Filter .................................. 27
Morphological Reconstruction .......... 38

N
Non-Local Means Denoising ............... 28

T
the GNU Project .............................. 1

U
User Configuration ........................... 43
Procedure Index

This is an alphabetical list of all the procedures, methods and macros in Guile-CV.

D
degree->radian ........................................ 44

I
im-=channel? ........................................... 12
im-=? ................................................. 12
im-add .................................................. 29
im-add-channel ........................................ 29
im-and ................................................. 31
im-and-channel ......................................... 31
im-binary-channel? ..................................... 12
im-binary? ............................................ 12
im-canny .............................................. 39
im-canny-channel ...................................... 39
im-channel ............................................. 11
im-channel-offset ...................................... 12
im-channel-ref ......................................... 12
im-channel-set! ......................................... 13
im-channels ............................................ 11
im-clip .................................................. 33
im-clip-channel ........................................ 33
im-close ............................................... 37
im-close-channel ........................................ 37
im-collect ............................................. 13
im-complement ......................................... 33
im-compose .............................................. 40
im-compose-channels .................................... 40
im-convolve ........................................... 27
im-convolve-channel .................................... 27
im-copy ............................................... 11
im-copy-channel ......................................... 11
im-crack-edge .......................................... 40
im-crack-edge-channel .................................. 40
im-crop ................................................. 33
im-crop-channel ........................................ 33
im-crop-size ........................................... 34
im-delineate .......................................... 37
im-delineate-channel ................................... 37
im-disc-dilate .......................................... 36
im-disc-dilate-channel .................................. 36
im-disc-erode .......................................... 35
im-disc-erode-channel .................................. 35
im-display ............................................... 41
im-display-channel ....................................... 41
im-distance-map ........................................ 38
im-distance-map-channel ................................ 38
im-divide ............................................... 30
im-divide-channel ....................................... 30
im-fast-channel-offset .................................. 12
im-fast-channel-ref ..................................... 12
im-fast-channel-set! ..................................... 13
<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>im-n-channel</td>
<td>11, 20</td>
</tr>
<tr>
<td>im-nl-means</td>
<td>28</td>
</tr>
<tr>
<td>im-nl-means-channel</td>
<td>28</td>
</tr>
<tr>
<td>im-normalize</td>
<td>31</td>
</tr>
<tr>
<td>im-normalize-channel</td>
<td>31</td>
</tr>
<tr>
<td>im-open</td>
<td>37</td>
</tr>
<tr>
<td>im-open-channel</td>
<td>37</td>
</tr>
<tr>
<td>im-or</td>
<td>31</td>
</tr>
<tr>
<td>im-or-channel</td>
<td>31</td>
</tr>
<tr>
<td>im-padd</td>
<td>34</td>
</tr>
<tr>
<td>im-padd-channel</td>
<td>34</td>
</tr>
<tr>
<td>im-padd-size</td>
<td>34</td>
</tr>
<tr>
<td>im-particle-clean</td>
<td>26</td>
</tr>
<tr>
<td>im-particles</td>
<td>26</td>
</tr>
<tr>
<td>im-range</td>
<td>30</td>
</tr>
<tr>
<td>im-range-channel</td>
<td>30</td>
</tr>
<tr>
<td>im-reconstruct</td>
<td>38</td>
</tr>
<tr>
<td>im-reduce</td>
<td>31</td>
</tr>
<tr>
<td>im-reduce-channel</td>
<td>31</td>
</tr>
<tr>
<td>im-ref</td>
<td>12</td>
</tr>
<tr>
<td>im-resize</td>
<td>32</td>
</tr>
<tr>
<td>im-resize-channel</td>
<td>32</td>
</tr>
<tr>
<td>im-rgb-&gt;gray</td>
<td>32</td>
</tr>
<tr>
<td>im-rgb?</td>
<td>11, 20</td>
</tr>
<tr>
<td>im-rgb-&gt;gray</td>
<td>32</td>
</tr>
<tr>
<td>im-rgb-&gt;rgb</td>
<td>32</td>
</tr>
<tr>
<td>im-rotate</td>
<td>32</td>
</tr>
<tr>
<td>im-rotate-channel</td>
<td>32</td>
</tr>
<tr>
<td>im-save</td>
<td>19</td>
</tr>
<tr>
<td>im-scrap</td>
<td>29</td>
</tr>
<tr>
<td>im-set!</td>
<td>12</td>
</tr>
<tr>
<td>im-sharpen</td>
<td>27</td>
</tr>
<tr>
<td>im-sharpen-channel</td>
<td>27</td>
</tr>
<tr>
<td>im-show</td>
<td>42</td>
</tr>
<tr>
<td>im-size</td>
<td>11, 20</td>
</tr>
<tr>
<td>im-subtract</td>
<td>29</td>
</tr>
<tr>
<td>im-subtract-channel</td>
<td>29</td>
</tr>
<tr>
<td>im-texture</td>
<td>21</td>
</tr>
<tr>
<td>im-threshold</td>
<td>29</td>
</tr>
<tr>
<td>im-times</td>
<td>30</td>
</tr>
<tr>
<td>im-times-channel</td>
<td>30</td>
</tr>
<tr>
<td>im-transpose</td>
<td>33</td>
</tr>
<tr>
<td>im-transpose-channel</td>
<td>33</td>
</tr>
<tr>
<td>im-width-channel</td>
<td>11, 20</td>
</tr>
<tr>
<td>im-xor</td>
<td>31</td>
</tr>
<tr>
<td>im-xor-channel</td>
<td>31</td>
</tr>
<tr>
<td>im-xor-channel</td>
<td>31</td>
</tr>
<tr>
<td>im-xor-channel</td>
<td>31</td>
</tr>
</tbody>
</table>

**K**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-channel</td>
<td>16</td>
</tr>
<tr>
<td>k-display</td>
<td>16</td>
</tr>
<tr>
<td>k-fast-offset</td>
<td>16</td>
</tr>
<tr>
<td>k-fast-ref</td>
<td>16</td>
</tr>
<tr>
<td>k-fast-set!</td>
<td>16</td>
</tr>
<tr>
<td>k-height</td>
<td>16</td>
</tr>
<tr>
<td>k-make</td>
<td>14</td>
</tr>
<tr>
<td>k-make-circular-mask</td>
<td>15</td>
</tr>
<tr>
<td>k-offset</td>
<td>16</td>
</tr>
<tr>
<td>k-ref</td>
<td>16</td>
</tr>
<tr>
<td>k-set!</td>
<td>16</td>
</tr>
<tr>
<td>k-size</td>
<td>16</td>
</tr>
<tr>
<td>k-width</td>
<td>16</td>
</tr>
<tr>
<td>kernel?</td>
<td>16</td>
</tr>
</tbody>
</table>

**R**

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>radian-&gt;degree</td>
<td>44</td>
</tr>
<tr>
<td>re-export-public-interface</td>
<td>44</td>
</tr>
</tbody>
</table>
Variable Index

This is an alphabetical list of all the important variables and constants in Guile-CV.

%2pi ................................................. 44  \%k-mean ........................................... 17
%image-cache ....................................... 43  \%k-prewitt-x ...................................... 19
%image-cache-format ............................... 43  \%k-prewitt-y ...................................... 18
\%k-edge0 ......................................... 17  \%k-sharpen ...................................... 17
\%k-edge1 ......................................... 17  \%k-sobel-x ........................................ 19
\%k-emboss ....................................... 18  \%k-sobel-y ....................................... 19
\%k-gaussian-blur0 ............................... 17  \%k-unsharp ........................................ 18
\%k-gaussian-blur1 ............................... 18  \%pi .............................................. 44
\%k-identity ..................................... 17  \%pi/2 ........................................... 44
\%k-laplacian .................................... 18
Type Index

This is an alphabetical list of all the important data types defined in the Guile-CV Programmers Manual.

(Index is nonexistent)