# G'MIC

GREYC's Magic for Image Computing

A Full-Featured Open-Source Framework for Image Processing

# The Handbook

**Version 3.3.5**

2024/3/12

# Preamble

- This document is distributed under the **GNU Free Documentation License**, version 1.3.
- A **.pdf version** of this document is available.

# Version

**G'MIC**: `GREYC's Magic for Image Computing`
**https://gmic.eu**
Version **3.3.5**

# Table of Contents

# Usage

```
gmic [command1 [arg1_1,arg1_2,..]] .. [commandN [argN_1,argN_2,..]]
```

`gmic` is the open-source interpreter of the **G'MIC** language, a scripting programming language dedicated to the design of possibly complex image processing pipelines and operators.
It can be used to convert, manipulate, filter and visualize image datasets made of one or several 1D/2D or 3D multi-spectral images.

This reference documentation describes all the technical aspects of the G'MIC framework, in its current version ***3.3.5***.

As a starting point, you may want to visit our detailed tutorial pages, at: **https://gmic.eu/tutorial/**

# Overall Context

- At any time, **G'MIC** manages one list of numbered (and optionally named) pixel-based images, entirely stored in computer memory (uncompressed).
- The first image of the list has index `0` and is denoted by `[0]`. The second image of the list is

denoted by `[1]`, the third by `[2]` and so on.

- Negative indices are treated in a periodic way: `[-1]` refers to the last image of the list, `[-2]` to the penultimate one, etc. Thus, if the list has 4 images, `[1]` and `[-3]` both designate the second image of the list.
- A named image may be also indicated by `[name]`, if `name` uses the character set `[a-zA-Z0-9_]` and does not start with a number. Image names can be set or reassigned at any moment during the processing pipeline (see command **name** for this purpose).
- G'MIC defines a set of various commands and substitution mechanisms to allow the design of complex pipelines and operators managing this list of images, in a very flexible way: You can insert or remove images in the list, rearrange image order, process images (individually or grouped), merge image data together, display and output image files, etc.
- Such a pipeline can define a new custom G'MIC command (stored in a user command file), and re-used afterwards as a regular command, in a larger pipeline if necessary.

# Image Definition and Terminology

- In **G'MIC**, each image is modeled as a 1D, 2D, 3D or 4D array of scalar values, uniformly discretized on a rectangular/parallelepipedic domain.
- The four dimensions of this array are respectively denoted by:
    - `width`, the number of image columns (size along the `x-axis`).
    - `height`, the number of image rows (size along the `y-axis`).
    - `depth`, the number of image slices (size along the `z-axis`). The depth is equal to `1` for usual color or grayscale 2D images.
    - `spectrum`, the number of image channels (size along the `c-axis`). The spectrum is respectively equal to `3` and `4` for usual `RGB` and `RGBA` color images.
- There are no hard limitations on the size of the image along each dimension. For instance, the number of image slices or channels can be of arbitrary size within the limits of the available memory.
- The `width`, `height` and `depth` of an image are considered as *spatial* dimensions, while the `spectrum` has a *multi-spectral* meaning. Thus, a 4D image in G'MIC should be most often regarded as a 3D dataset of multi-spectral voxels. Most of the G'MIC commands will stick with this idea (e.g. command **blur** blurs images only along the spatial `xyz`-axes).
- G'MIC stores all the image data as buffers of `float` values (32 bits, value range `[-3.4E38,+3.4E38]`. It performs all its image processing operations with floating point numbers. Each image pixel takes then 32bits/channel (except if double-precision buffers have been enabled during the compilation of the software, in which case 64bits/channel can be the default).
- Considering `float`-valued pixels ensure to keep numerical precision when executing image processing pipelines. For image input/output operations, you may want to prescribe the image datatype to be different than `float` (like `bool`, `char`, `int`, etc.). This is possible by specifying it as a file option when using I/O commands (see section **Output Properties** to learn more about file options).

# Items of a Processing Pipeline

- In **G'MIC**, an image processing pipeline is described as a sequence of items separated by the **space** character. Such items are interpreted and executed from the left to the right. For instance, the expression:

```
filename.jpg blur 3,0 sharpen 10 resize 200%,200% output file_out.jpg
```

defines a valid pipeline composed of nine G'MIC items.

- Each G'MIC item is either a **command**, a list of **command arguments**, a **filename** or a special **input string**.
- Escape characters " and double quotes `"` can be used to define items containing spaces or other special characters. For instance, the two strings `single\ item` and `"single item"` both define the same single item, with a space in it.

# Input Data

- If a specified **G'MIC** item appears to be an existing filename, the corresponding image data are loaded and inserted at the end of the image list (which is equivalent to the use of `input filename`).
- Special filenames `-` and `-.ext` stand for the standard input/output streams, optionally forced to be in a specific `ext` file format (e.g. `-.jpg` or `-.png`).
- The following special input strings may be used as G'MIC items to create and insert new images with prescribed values, at the end of the image list:
  - `[selection]` or `[selection]xN`: Insert 1 or N copies of already existing images. `selection` may represent one or several images (see section **Command Items and Selections** to learn more about selections).
  - `width[%],_height[%],_depth[%],_spectrum[%],_values[xN]`: Insert one or N images with specified size and values (adding `%` to a dimension means **"percentage of the size along the same axis"**, taken from the last image `[-1]`). Any specified dimension can be also written as `[image]`, and is then set to the size (along the same axis) of the existing specified image `[image]`. `values` can be either a sequence of numbers separated by commas `,`, or a mathematical expression, as e.g. in input item `256,256,1,3,[x,y,128]` which creates a `256x256` RGB color image with a spatial shading on the red and green channels. (see section **Mathematical Expressions** to learn more about mathematical expressions).
  - `(v1,v2,..)[xN]`: Insert one or `N` new images from specified prescribed values. Value separator inside parentheses can be `,` (column separator), `;` (row separator), `/` (slice separator) or `^` (channel separator). For instance, expression `(1,2,3;4,5,6;7,8,9)` creates a 3x3 matrix (scalar image), with values running from 1 to 9.
  - `('string'[:delimiter])[xN]`: Insert one or N new images from specified string, by filling the images with the character codes composing the string. When specified, `delimiter` tells about the main orientation of the image. Delimiter can be `x` (eq. to `,` which is the default), `y` (eq. to `;`), `z` (eq. to `/`) or `c` (eq. to `^`). When specified delimiter is `,`, `;`, `/` or `^`, the expression is actually equivalent to `({'string'[:delimiter]})[xN]` (see section **Substitution Rules** for more information on the syntax).

  - `0[xN]` : Insert one or N new `empty` images, containing no pixel data. Empty images are used only in rare occasions.

- Input item `name=value` declares a new variable `name`, or assign a new string value to an existing variable. Variable names must use the character set `[a-zA-Z0-9_]` and cannot start with a number.
- A variable definition is always local to the current command except :
  - When it starts by the underscore character `_`. In that case, it becomes also accessible by any command invoked outside the current command scope (global variable).
  - When defined in a *subcommand* of the current command, a variable becomes also accessible in the parent command. A *subcommand* of a command `foo` is a command whose name starts with `_foo` (e.g. `_foo_sub`) and that is called from `foo`.
- If a variable name starts with two underscores `__`, the global variable is also shared among different threads and can be read/set by commands running in parallel (see command **parallel** for this purpose). Otherwise, it remains local to the thread that defined it.
- Numerical variables can be updated with the use of these special operators: `+=` (addition), `-=` (subtraction), `*=` (multiplication), `/=` (division), `%=` (modulo), `&=` (bitwise and), `|=` (bitwise or), `^=` (power), `<<=` and `>>` (bitwise left and right shifts). For instance, `foo=1` `foo+=3`.
- Input item `name.=string` appends specified `string` at the end of variable `name`.
- Input item `name..=string` prepends specified `string` at the beginning of variable `name`.
- Multiple variable assignments and updates are allowed, with expressions: `name1,name2,...,nameN=value` or `name1,name2,...,nameN=value1,value2,...,valueN` where assignment operator `=` can be replaced by one of the allowed operators (e.g. `+=`).
- Variables usually store numbers or strings. Use command **store** to assign variables from image data (and syntax `input $variable` to bring them back on the image list afterwards).

# Command Items and Selections

- A **G'MIC** item that is not a filename nor a special input string designates a `command` most of the time. Generally, commands perform image processing operations on one or several available images of the list.
- Reccurent commands have two equivalent names (`regular` and `short`). For instance, command names `resize` and `r` refer to the same image resizing action.
- A G'MIC command may have mandatory or optional **arguments**. Command arguments must be specified in the next item on the command line. Commas `,` are used to separate multiple arguments of a single command, when required.
- The execution of a G'MIC command may be restricted only to a **subset** of the image list, by appending `[selection]` to the command name. Examples of valid syntaxes for `selection` are:
  - `command[-2]` : Apply command only on the penultimate image `[-2]` of the list.
  - `command[0,1,3]` : Apply command only on images `[0]`, `[1]` and `[3]`.

    - `command[3-6]`: Apply command only on images `[3]` to `[6]` (i.e, `[3]`, `[4]`, `[5]` and `[6]`).
    - `command[50%-100%]`: Apply command only on the second half of the image list.
    - `command[0,-4--1]`: Apply command only on the first image and the last four images.
    - `command[0-9:3]`: Apply command only on images `[0]` to `[9]`, with a step of 3 (i.e. on images `[0]`, `[3]`, `[6]` and `[9]`).
    - `command[0--1:2]`: Apply command only on images of the list with even indices.
    - `command[0,2-4,50%--1]`: Apply command on images `[0]`, `[2]`, `[3]`, `[4]` and on the second half of the image list.
    - `command[^0,1]`: Apply command on all images except the first two.
    - `command[name1,name2]`: Apply command on named images `name1` and `name2`.

- Indices in selections are always sorted in increasing order, and duplicate indices are discarded. For instance, selections `[3-1,1-3]` and `[1,1,1,3,2]` are both equivalent to `[1-3]`. If you want to repeat a single command multiple times on an image, use a `repeat..done` loop instead. Inverting the order of images for a command is achieved by explicitly inverting the order of the images in the list, with command `reverse[selection]`.
- Command selections `[-1]`, `[-2]` and `[-3]` are so often used they have their own shortcuts, respectively `.`, `..` and `...`. For instance, command `blur..` is equivalent to `blur[-2]`. These shortcuts work also when specifying command arguments.
- G'MIC commands invoked without `[selection]` are applied on all images of the list, i.e. the default selection is `[0--1]` (except for command **input** whose default selection is `[-1]'`).
- Prepending a single hyphen `-` to a G'MIC command is allowed. This may be useful to recognize command items more easily in a one-liner pipeline (typically invoked from a shell).
- A G'MIC command prepended with a plus sign `+` does not act **in-place** but inserts its result as one or several new images at the end of the image list.
- There are two different types of commands that can be run by the G'MIC interpreter:
    - **Built-in commands** are the hard-coded functionalities in the interpreter core. They are thus compiled as binary code and run fast, most of the time. Omitting an argument when invoking a built-in command is not permitted, except if all following arguments are also omitted. For instance, invoking 'plasma 10,,5' is invalid but 'plasma 10' is correct.
    - **Custom commands**, are defined as G'MIC pipelines of built-in or other custom commands. They are parsed by the G'MIC interpreter, and thus run a bit slower than built-in commands. Omitting arguments when invoking a custom command is permitted. For instance, expressions `flower ,,,100,,2` or `flower ,` are correct.
- Most of the existing commands in G'MIC are actually defined as **custom commands**.
- A user can easily add its own custom commands to the G'MIC interpreter (see section **Adding Custom Commands** for more details). New built-in commands cannot be added (unless you modify the G'MIC interpreter source code and recompile it).

# Input/Output Properties

- G'MIC is able to read/write most of the classical image file formats, including:
    - 2D grayscale/color files: `.png`, `.jpeg`, `.gif`, `.pnm`, `.tif`, `.bmp`, ...
    - 3D volumetric files: `.dcm`, `.hdr`, `.nii`, `.cube`, `.pan`, `.inr`, `.pnk`, ...
    - Video files: `.mpeg`, `.avi`, `.mp4`, `.mov`, `.ogg`, `.flv`, ...
    - Generic text or binary data files: `.gmz`, `.cimg`, `.cimgz`, `flo`, `ggr`, `gpl`, `.dlm`, `.asc`, `.pfm`, `.raw`, `.txt`, `.h`.
    - 3D mesh files: `.off`, `.obj`.

- When dealing with color images, G'MIC generally reads, writes and displays data using the usual sRGB color space.
- When loading a `.png` and `.tiff` file, the bit-depth of the input image(s) is returned to the status.
- G'MIC is able to manage **3D mesh objects** that may be read from files or generated by G'MIC commands. A 3D object is stored as a one-column scalar image containing the object data, in the following order: ***{ magic_number; sizes; vertices; primitives; colors; opacities }***. These 3D representations can be then processed as regular images (see command **split3d** for accessing each of these 3D object data separately).
- Be aware that usual file formats may be sometimes not adapted to store all the available image data, since G'MIC uses float-valued image buffers. For instance, saving an image that was initially loaded as a 16bits/channel image, as a `.jpg` file will result in a loss of information. Use the G'MIC-specific file extension `.gmz` to ensure that all data precision is preserved when saving images.
- Sometimes, file options may/must be set for file formats:
    - **Video files:** Only sub-frames of an image sequence may be loaded, using the input expression `filename.ext,[first_frame[,last_frame[,step]]]`. Set `last_frame==-1` to tell it must be the last frame of the video. Set `step` to `0` to force an opened video file to be opened/closed. Output framerate and codec can be also set by using the output expression `filename.avi,_fps,_codec,_keep_open` where `keep_open` can be ***{ 0 | 1 }***. `codec` is a 4-char string (see **http://www.fourcc.org/codecs.php** ) or `0` for the default codec. `keep_open` tells if the output video file must be kept open for appending new frames afterwards.
    - `.cimg[z]` **files:** Only crops and sub-images of .cimg files can be loaded, using the input expressions `filename.cimg,N0,N1`, `filename.cimg,N0,N1,x0,x1`, `filename.cimg,N0,N1,x0,y0,x1,y1`, `filename.cimg,N0,N1,x0,y0,z0,x1,y1,z1` or `filename.cimg,N0,N1,x0,y0,z0,c0,x1,y1,z1,c1`. Specifying `-1` for one coordinates stands for the maximum possible value. Output expression `filename.cimg[z][,datatype]` can be used to force the output pixel type. `datatype` can be ***{ auto | bool | uint8 | int8 | uint16 | int16 | uint32 | int32 | uint64 | int64 | float32 | float64 }***.
    - `.raw` **binary files:** Image dimensions and input pixel type may be specified when loading `.raw` files with input expression `filename.raw[,datatype][,width][,height[,depth[,dim[,offset]]]]]`. If no dimensions are specified, the resulting image is a one-column vector with maximum possible height. Pixel type can also be specified with the output expression `filename.raw[,datatype]`. `datatype` can be the same as for `.cimg[z]` files.
    - `.yuv` **files:** Image dimensions must be specified when loading, and only sub-frames of an

    image sequence may be loaded, using the input expression `filename.yuv,width,height[,chroma_subsampling[,first_frame[,last_frame[,ste` `chroma_subsampling` can be ***`{ 420 | 422 | 444 }`***. When saving, chroma subsampling mode can be specified with output expression `filename.yuv[,chroma_subsampling]`.

  - `.tiff` **files:** Only sub-images of multi-pages tiff files can be loaded, using the input expression `filename.tif,_first_frame,_last_frame,_step`. Output expression `filename.tiff,_datatype,_compression,_force_multipage,_use_bigtiff` can be used to specify the output pixel type, as well as the compression method. `datatype` can be the same as for `.cimg[z]` files. `compression` can be ***`{ none (default) | lzw | jpeg }`***. `force_multipage` can be ***`{ 0:no (default) | 1:yes }`***. `use_bigtiff` can be ***`{ 0:no | 1:yes (default) }`***.
  - `.pdf` **files:** When loading a file, the rendering resolution can be specified using the input expression `filename.pdf,resolution`, where `resolution` is an unsigned integer value.
  - `.gif` **files:** Animated gif files can be saved, using the input expression `filename.gif,fps>0,nb_loops`. Specify `nb_loops=0` to get an infinite number of animation loops (this is the default behavior).
  - `.jpeg` **files:** The output quality may be specified (in %), using the output expression `filename.jpg,30` (here, to get a 30% quality output). `100` is the default.
  - `.mnc` **files:** The output header can set from another file, using the output expression `filename.mnc,header_template.mnc`.
  - `.pan`, `.cpp`, `.hpp`, `.c` and `.h` **files:** The output datatype can be selected with output expression `filename[,datatype]`. `datatype` can be the same as for `.cimg[z]` files.
  - `.gmic` **files:** These filenames are assumed to be G'MIC custom commands files. Loading such a file will add the commands it defines to the interpreter. Debug information can be enabled/disabled by the input expression `filename.gmic[,add_debug_info]` where `debug_info` can be ***`{ 0:false | 1:true }`***.
  - Inserting `ext:` on the beginning of a filename (e.g. `jpg:filename`) forces G'MIC to read/write the file as it would have been done if it had the specified extension `.ext`.

- Some input/output formats and options may not be supported, depending on the configuration flags that have been set during the build of the G'MIC software.

# Substitution Rules

- **G'MIC** items containing `$` or `{}` are substituted before being interpreted. Use these substituting expressions to access various data from the interpreter environment.
- `$name` and `${name}` are both substituted by the value of the specified named variable (set previously by the item `name=value`). If this variable has not been already set, the expression is substituted by the highest positive index of the named image `[name]`. If no image has this name, the expression is substituted by the value of the OS environment variable with same name (it may be thus an empty string if it is not defined).
- The following reserved variables are predefined by the G'MIC interpreter:
  - `$!`: The current number of images in the list.
  - `$>` and `$<`: The increasing/decreasing index of the latest (currently running) `repeat...done` loop. `$>` goes from `0` (first loop iteration) to `nb_iterations - 1` (last

iteration). `$<` does the opposite.
  - `$/` : The current call stack. Stack items are separated by slashes `/` .
  - `$|` : The current value (expressed in seconds) of a millisecond precision timer.
  - `$^` : The current verbosity level.
  - `$_cpus` : The number of computation cores available on your machine.
  - `$_flags` : The list of enabled flags when G'MIC interpreter has been compiled.
  - `$_host` : A string telling about the host running the G'MIC interpreter (e.g. `cli` or `gimp`).
  - `$_os` : A string describing the running operating system.
  - `$_path_rc` : The path to the G'MIC folder used to store configuration files (its value is OS-dependent).
  - `$_path_user` : The path to the G'MIC user file `.gmic` or `user.gmic` (its value is OS-dependent).
  - `$_path_commands` : A list of all imported command files (stored as an image list).
  - `$_pid` : The current process identifier, as an integer.
  - `$_pixeltype` : The type of image pixels (default: `float32` ).
  - `$_prerelease` : For pre-releases, the date of the pre-release as `yymmdd`. For stable releases, this variable is set to `0`.
  - `$_version` : A 3-digits number telling about the current version of the G'MIC interpreter (e.g. `335` ).
  - `$_vt100` : Set to `1` if colored text output is allowed on the console. Otherwise, set to `0`.

- `$$name` and `$${name}` are both substituted by the G'MIC script code of the specified named `custom command`, or by an empty string if no custom command with specified name exists.
- `${"-pipeline"}` is substituted by the **status value** after the execution of the specified G'MIC pipeline (see command **status**). Expression `${}` thus stands for the current status value.
- `{``string}` (starting with two backquotes) is substituted by a double-quoted version of the specified string.
- `{/string}` is substituted by the escaped version of the specified string.
- `{'string'[:delimiter]}` (between single quotes) is substituted by the sequence of character codes that composes the specified string, separated by specified delimiter. Possible delimiters are `,` (default), `;` , `/` , `^` or ' '. For instance, item `{'foo'}` is substituted by `102,111,111` and `{'foo` :;}' by `102;111;111` .
- `{image,feature[:delimiter]}` is substituted by a specific feature of the image `[image]` . `image` can be either an image number or an image name. It can be also eluded, in which case, the last image `[-1]` of the list is considered for the requested feature. Specified `feature` can be one of:
  - `b` : The image basename (i.e. filename without the folder path nor extension).
  - `f` : The image folder name.
  - `n` : The image name or filename (if the image has been read from a file).
  - `t` : The text string from the image values regarded as character codes.
  - `x` : The image extension (i.e the characters after the last `.` in the image name).

  - `^`: The sequence of all image values, separated by commas `,`.
  - `@subset`: The sequence of image values corresponding to the specified subset, and separated by commas `,`.
  - Any other `feature` is considered as a **mathematical expression** associated to the image `[image]` and is substituted by the result of its evaluation (float value). For instance, expression `{0,w+h}` is substituted by the sum of the width and height of the first image (see section **Mathematical Expressions** for more details). If a mathematical expression starts with an underscore `_`, the resulting value is truncated to a readable format. For instance, item `{_pi}` is substituted by `3.14159` (while `{pi}` is substituted by `3.141592653589793`).
  - A `feature` delimited by backquotes is replaced by a string whose character codes correspond to the list of values resulting from the evaluation of the specified mathematical expression. For instance, item ``{`[102,111,111]`}`` is substituted by `foo` and item ``{`vector8(65)`}`` by `AAAAAAAA`.

- `{*}` is substituted by the visibility state of the instant display window `#0` (can be ***{ 0:closed | 1:visible }***.
- `{*[index],feature1,...,featureN[:delimiter]}` is substituted by a specific set of features of the instant display window `#0` (or `#index`, if specified). Requested `features` can be:
  - `u`: screen width (actually independent on the window size).
  - `v`: screen height (actually independent on the window size).
  - `uv`: screen width*screen height.
  - `d`: window width (i.e. width of the window widget).
  - `e`: window height (i.e. height of the window widget).
  - `de`: window width*window height.
  - `w`: display width (i.e. width of the display area managed by the window).
  - `h`: display height (i.e. height of the display area managed by the window).
  - `wh`: display width*display height.
  - `i`: X-coordinate of the display window.
  - `j`: Y-coordinate of the display window.
  - `f`: current fullscreen state of the instant display.
  - `n`: current normalization type of the instant display.
  - `t`: window title of the instant display.
  - `x`: X-coordinate of the mouse position (or -1, if outside the display area).
  - `y`: Y-coordinate of the mouse position (or -1, if outside the display area).
  - `b`: state of the mouse buttons ***{ 1:left-but. | 2:right-but. | 4:middle-but. }***.
  - `o`: state of the mouse wheel.
  - `k`: decimal code of the pressed key if any, 0 otherwise.
  - `c`: boolean (0 or 1) telling if the instant display has been closed recently.
  - `r`: boolean telling if the instant display has been resized recently.
  - `m`: boolean telling if the instant display has been moved recently.

  - Any other `feature` stands for a keycode name (in capital letters), and is substituted by a boolean describing the current key state *`{ 0:pressed | 1:released }`*.
  - You can also prepend a hyphen `-` to a `feature` (that supports it) to flush the corresponding event immediately after reading its state (works for keys, mouse and window events).

- Item substitution is **never** performed in items between double quotes. One must break the quotes to enable substitution if needed, as in '"3+8 kg = "{3+8}" kg"'. Using double quotes is then a convenient way to disable the substitutions mechanism in items, when necessary.
- One can also disable the substitution mechanism on items outside double quotes, by escaping the `{`, `}` or `$` characters, as in `\{3+4\}\ doesn't\ evaluate`.

# Mathematical Expressions

- **G'MIC** has an embedded **mathematical parser**, used to evaluate (possibly complex) math expressions specified inside braces `{}`, or formulas in commands that may take one as an argument (e.g. **fill** or **eval**).
- When the context allows it, a formula is evaluated **for each pixel** of the selected images (e.g. **fill** or **eval**).
- A math expression may return or take as an argument a **scalar** or a **vector-valued** result (with a fixed number of components).

The mathematical parser understands the following set of functions, operators and variables:

## Usual math operators:

`||` (logical or), `&&` (logical and), `|` (bitwise or), `&` (bitwise and), `!=`, `==`, `<=`, `>=`, `<`, `>`, `<<` (left bitwise shift), `>>` (right bitwise shift), `-`, `+`, `*`, `/`, `%` (modulo), `^` (power), `!` (logical not), `~` (bitwise not), `++`, `--`, `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `>>`, `<<=` (in-place operators).

## Usual math functions:

`abs()`, `acos()`, `acosh()`, `arg()`, `arg0()`, `argkth()`, `argmax()`, `argmaxabs()`, `argmin()`, `argminabs()`, `asin()`, `asinh()`, `atan()`, `atan2()`, `atanh()`, `avg()`, `bool()`, `cbrt()`, `ceil()`, `cos()`, `cosh()`, `cut()`, `deg2rad()`, `erf()`, `erfinv()`, `exp()`, `fact()`, `fibo()`, `floor()`, `gamma()`, `gauss()`, `gcd()`, `hypot()`, `int()`, `isconst()`, `isnan()`, `isnum()`, `isinf()`, `isint()`, `isbool()`, `isexpr()`, `isfile()`, `isdir()`, `isin()`, `kth()`, `lcm()`, `log()`, `log2()`, `log10()`, `max()`, `maxabs()`, `med()`, `min()`, `minabs()`, `narg()`, `prod()`, `rad2deg()`, `rol()` (left bit rotation), `ror()` (right bit rotation), `round()`, `sign()`, `sin()`, `sinc()`, `sinh()`, `sqrt()`, `std()`, `srand(_seed)`, `sum()`, `tan()`, `tanh()`, `var()`, `xor()`.

- `cov(A,B,_avgA,_avgB)` estimates the covariance between vectors `A` and `B` (estimated averages of these vectors may be specified as arguments).
- `mse(A,B)` returns the mean-squared error between vectors `A` and `B`.
- `atan2(y,x)` is the version of `atan()` with two arguments `y` and `x` (as in C/C++).

- `perm(k,n,_with_order)` computes the number of permutations of `k` objects from a set of `n` objects.

- `gauss(x,_sigma,_is_normalized)` returns `exp(-x^2/(2*s^2))/(is_normalized?sqrt(2*pi*sigma^2):1)`.

- `cut(value,min,max)` returns `value` if it is in range `[min,max]`, or `min` or `max` otherwise.

- `narg(a_1,...,a_N)` returns the number of specified arguments (here, `N`).

- `arg(i,a_1,..,a_N)` returns the `i`-th argument `a_i`.

- `isnum()`, `isnan()`, `isinf()`, `isint()`, `isbool()` test the type of the given number or expression, and return `0` (false) or `1` (true).

- `isfile('path')` (resp. `isdir('path`)') returns `0` (false) or `1` (true) whether its string argument is a path to an existing file (resp. to a directory) or not.

- `ispercentage(arg)` returns `1` (true) or `0` (false) whether `arg` ends with a `%` or not.

- `isvarname('str')` returns `0` (false) or `1` (true) whether its string argument would be a valid to name a variable or not.

- `isin(v,a_1,...,a_n)` returns `0` (false) or `1` (true) whether the first argument `v` appears in the set of other argument `a_i`.

- `isint(x,_xmin,_xmax)` returns `1` (true), if `x` is an integer in range `[xmin,xmax]`, otherwise `0` (false).

- `inrange(value,m,M,include_m,include_M)` returns `0` (false) or `1` (true) whether the specified value lies in range `[m,M]` or not (`include_m` and `includeM` tells how boundaries `m` and `M` are considered).

- `argkth()`, `argmin()`, `argmax()`, `argminabs()`, `argmaxabs()'`, `avg()`, `kth()`, `min()`, `max()`, `minabs()`, `maxabs()`, `med()`, `prod()`, `std()`, `sum()` and `var()` can be called with an arbitrary number of scalar/vector arguments.

- `vargkth()`, `vargmin()`, `vargmax()`, `vargminabs()`, `vargmaxabs()`, `vavg()`, `vkth()`, `vmin()`, `vmax()`, `vminabs()`, `vmaxabs()`, `vmed()`, `vprod()`, `vstd()`, `vsum()` and `vvar()` are the versions of the previous function with vector-valued arguments.

- `round(value,rounding_value,direction)` returns a rounded value. `direction` can be ***{ -1:to-lowest | 0:to-nearest | 1:to-highest }***.

- `lerp(a,b,t)` returns `a*(1-t)+b*t`.

- `swap(a,b)` swaps the values of the given arguments.

# Predefined variable names:

Variable names below are pre-defined. They can be overridden though.

- `l`: length of the associated list of images.

- `k`: index of the associated image, in `[0,l-1]`.

- `w`: width of the associated image, if any (`0` otherwise).

- `h`: height of the associated image, if any (`0` otherwise).

- `d`: depth of the associated image, if any (`0` otherwise).

- `s` : spectrum of the associated image, if any (`0` otherwise).
- `r` : shared state of the associated image, if any (`0` otherwise).
- `wh` : shortcut for `width*height`.
- `whd` : shortcut for `width*height*depth`.
- `whds` : shortcut for `width*height*depth*spectrum` (i.e. number of image values).
- `im`, `iM`, `ia`, `iv`, `id`, `is`, `ip`, `ic`, `in` : Respectively the minimum, maximum, average, variance, standard deviation, sum, product, median value and L2-norm of the associated image, if any (`0` otherwise).
- `xm`, `ym`, `zm`, `cm` : The pixel coordinates of the minimum value in the associated image, if any (`0` otherwise).
- `xM`, `yM`, `zM`, `cM` : The pixel coordinates of the maximum value in the associated image, if any (`0` otherwise).
- All these variables are considered as **constant values** by the math parser (for optimization purposes) which is indeed the case most of the time. Anyway, this might not be the case, if function `resize(#ind,..)` is used in the math expression. If so, it is safer to invoke functions `l()`, `w(_#ind)`, `h(_#ind)`, ... `s(_#ind)` and `in(_#ind)` instead of the corresponding named variables.
- `i` : current processed pixel value (i.e. value located at `(x,y,z,c)`) in the associated image, if any (`0` otherwise).
- `iN` : N-th channel value of current processed pixel (i.e. value located at `(x,y,z,N)` in the associated image, if any (`0` otherwise). `N` must be an integer in range `[0,9]`.
- `R`, `G`, `B` and `A` are equivalent to `i0`, `i1`, `i2` and `i3` respectively.
- `I` : current vector-valued processed pixel in the associated image, if any (`0` otherwise). The number of vector components is equal to the number of image channels (e.g. `I` = `[ R,G,B ]` for a `RGB` image).
- You may add `#ind` to any of the variable name above to retrieve the information for any numbered image `[ind]` of the list (when this makes sense). For instance `ia#0` denotes the average value of the first image of the list).
- `x` : current processed column of the associated image, if any (`0` otherwise).
- `y` : current processed row of the associated image, if any (`0` otherwise).
- `z` : current processed slice of the associated image, if any (`0` otherwise).
- `c` : current processed channel of the associated image, if any (`0` otherwise).
- `t` : thread id when an expression is evaluated with multiple threads (`0` means **master thread**).
- `n` : maximum number of threads when expression is evaluated in parallel (so that `t` goes from `0` to `n-1`).
- `e` : value of e, i.e. `2.71828...`.
- `pi` : value of pi, i.e. `3.1415926...`.
- `eps` : value of machine epsilon, that is the difference between 1.0 and the next value representable by a double.
- `u` : a random value between `[0,1]`, following a uniform distribution.

- `g` : a random value, following a gaussian distribution of variance 1 (roughly in `[-6,6]`).
- `interpolation` : value of the default interpolation mode used when reading pixel values with the pixel access operators (i.e. when the interpolation argument is not explicitly specified, see below for more details on pixel access operators). Its initial default value is `0`.
- `boundary` : value of the default boundary conditions used when reading pixel values with the pixel access operators (i.e. when the boundary condition argument is not explicitly specified, see below for more details on pixel access operators). Its initial default value is `0`.
- The last image of the list is always associated to the evaluations of `expressions`, e.g. G'MIC sequence

  ```
  256,128 fill {w}
  ```

will create a 256x128 image filled with value 256.

# Vector-valued functions and operators:

The math evaluator is able to work with vector-valued elements. A math function applied on a vector-valued argument usually returns a vector with same dimension, where each element of the input vector has been passed to the specified function (e.g. `abs([-1,2,-3])` returns `[1,2,3]`).

There are specific functions and operators to define or compute vector-valued elements though :

- `[a0,a1,...,aN-1]` defines a `N`-dimensional vector with scalar coefficients `ak`.
- `vectorN(a0,a1,,...,aN-1)` does the same, with the `ak` being repeated periodically if only a few are specified.
- `vector(#N,a0,a1,,...,aN-1)` does the same, and can be used for any constant expression `N`.
- In previous expressions, the `ak` can be vectors themselves, to be concatenated into a single vector.
- The scalar element `ak` of a vector `X` is retrieved by `X[k]`.
- The sub-vector `[X[p],X[p+s]...X[p+s*(q-1)]]` (of size `q`) of a vector `X` is retrieved by `X[p,q,s]`.
- Equality/inequality comparisons between two vectors is done with operators `==` and `!=`.
- Some vector-specific functions can be used on vector values: `cross(X,Y)` (cross product), `dot(X,Y)` (dot product), `size(X)` (vector dimension), `sort(X,_is_increasing,_nb_elts,_size_elt)` (sorted values), `reverse(A)` (reverse order of components), `map(X,P,_nb_channelsX,_nb_channelsP,_boundary_conditions)`, `shift(A,_length,_boundary_conditions)` and `same(A,B,_nb_vals,_is_case_sensitive)` (vector equality test).
- Function `normP(u1,...,un)` computes the LP-norm of the specified vector (`P` being a constant or `inf`, as in e.g. `norm1()`).
- Function `normp(V,_p)` computes the Lp-norm of the specified vector `V`. Here, `p` can be variable. Default value for `p` is 2.
- Function `unitnorm(V,_p)` returns a normalized version `V/normp(V)` of specified vector `V`. Default value for `p` is 2.

- Function `resize(A,size,_interpolation,_boundary_conditions)` returns a resized version of a vector `A` with specified interpolation mode. `interpolation` can be ***{ -1:none (memory content) | 0:none | 1:nearest | 2:average | 3:linear | 4:grid | 5:bicubic | 6:lanczos }***, and `boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

- Function `find(A,B,_starting_index,_search_step)` returns the index where sub-vector `B` appears in vector `A`, (or `-1` if `B` is not contained in `A`). Argument `A` can be also replaced by an image index `#ind`.

- Specifying a vector-valued math expression as an argument of a command that operates on image values (e.g. `fill`) modifies the whole spectrum range of the processed image(s), for each spatial coordinates `(x,y,z)`. The command does not loop over the `c`-axis in this case.

# Complex-valued functions:

A `2`-dimensional vector may be seen as a complex number and used in those particular functions/ operators: `**` (complex multiplication), `//` (complex division), `^^` (complex exponentiation), `**=` (complex self-multiplication), `//=` (complex self-division), `^^=` (complex self-exponentiation), `cabs()` (complex modulus), `carg()` (complex argument), `cconj()` (complex conjugate), `cexp()` (complex exponential), `clog()` (complex logarithm), `ccos()` (complex cosine), `csin()` (complex sine), `csqr()` (complex square), `csqrt()` (complex square root), `ctan()` (complex tangent), `ccosh()` (complex hyperpolic cosine), `csinh()` (complex hyperbolic sine) and `ctanh()` (complex hyperbolic tangent).

# Matrix-valued functions:

A `MN`-dimensional vector may be seen as a `M` x `N` matrix and used in those particular functions/ operators: `*` (matrix-vector multiplication), `det(A)` (determinant), `diag(V)` (diagonal matrix from a vector), `eig(A)` (eigenvalues/eigenvectors), `eye(n)` (n x n identity matrix), `invert(A,_nb_colsA,_use_LU,_lambda)` (matrix inverse), `mul(A,B,_nb_colsB)` (matrix-matrix multiplication), `rot(u,v,w,angle)` (3D rotation matrix), `rot(angle)` (2D rotation matrix), `solve(A,B,_nb_colsB,_use_LU)` (solver of linear system A.X = B), `svd(A,_nb_colsA)` (singular value decomposition), `trace(A)` (matrix trace) and `transpose(A,nb_colsA)` (matrix transpose). Argument `nb_colsB` may be omitted if it is equal to `1`.

# Image-valued functions:

Some functions takes vector-valued arguments that represent image data :

- Function `expr(formula,_w,_h,_d,_s)` outputs a vector of size `w*h*d*s` with values generated from the specified formula, as if one were filling an image with dimensions `(w,h,d,s)`.

- Function `resize(A,wA,hA,dA,sA,nwA,_nhA,_ndA,_nsA,_interpolation,_boundary_conditions,_a` is an extended version of the `resize()` function. It allows to resize the vector `A`, seen as an image of size `(ow,oh,od,os)` as a new image of size `(nw,nh,nd,ns)`, with specified resizing options.

- Function `warp(A,wA,hA,dA,sA,B,wB,hB,dB,sB,_mode,_interpolation,_boundary_conditions)` returns the warped version of the image `A` (of size `(wA,hA,dA,sA)`, viewed as a vector of size `wA*hA*dA*sA`) by the warping field `B` (of size `(wB,hB,dB,sB)`). The resulting image has size `(wB,hB,dB,sA)`. This is the math evaluator analog to command **warp**.

- Function `index(A,P,nb_channelsP,_dithering,_map_colors)` returns the indexed version of the image `A` by the colormap `P`. This is the math evaluator analog to command **index**.
- Function `permute(A,wA,hA,dA,sA,permutation_string)` returns a permuted version of the image `A` (of size `(wA,hA,dA,sA)`, viewed as a vector of size `wA*hA*dA*sA`). This is the math evaluator analog to command **permute**.
- Function `mirror(A,wA,hA,dA,sA,axes_string)` returns a mirrored version of the image `A` (of size `(wA,hA,dA,sA)`, viewed as a vector of size `wA*hA*dA*sA`). This is the math evaluator analog to command **mirror**.
- Function `cumulate(A,wA,hA,dA,sA,_axes_string)` returns a cumulated version of the image `A` (of size `(wA,hA,dA,sA)`, viewed as a vector of size `wA*hA*dA*sA`). This is the math evaluator analog to command **cumulate**.
- Function `histogram(A,nb_levels,_min_value,_max_value)` returns the histogram of the vector `A`. This is the math evaluator analog to command **histogram**.
- Function `equalize(A,nb_levels,_min_value,_max_value)` returns the equalized version of the vector `A`. This is the math evaluator analog to command **equalize**.
- Function `normalize(A,_min_value,_max_value)` returns the normalized version of the vector `A`. This is the math evaluator analog to command **normalize**.
- `mproj(S,nb_colsS,D,nb_colsD,method,max_iter,max_residual)` projects a matrix `S` onto a dictionary (matrix) `D`. This is the math evaluator analog to command **mproj**.
- Function `noise(A,amplitude,_noise_type)` returns the noisy version of the vector `A`. This is the math evaluator analog to command **noise**.
- Function `rand(#size,_min_value,_max_value,_pdf,_precision)` returns the a vector of `size` random values. This is the math evaluator analog to command **rand**.

# String manipulation:

Character strings are defined as vectors objects and can be then managed as is. Dedicated functions and initializers to manage strings exist:

- `['string']` and `'string'` define a vector whose values are the character codes of the specified `character string` (e.g. `'foo'` is equal to `[ 102,111,111 ]`).
- `_'character'` returns the (scalar) byte code of the specified character (e.g. `_'A'` is equal to `65`).
- A special case happens for **empty** strings: Values of both expressions `['']` and `''` are `0`.
- Functions `lowercase()` and `uppercase()` return string with all string characters lowercased or uppercased.
- Function `s2v(str,_starting_index,_is_strict)` parses specified string `str` and returns the value contained in it.
- Function `v2s(expr,_nb_digits,_siz)` returns a vector of size `siz` which contains the character representation of values described by expression `expr`. `nb_digits` can be ***{ <-1:0-padding of integers | -1:auto-reduced | 0:all | >0:max number of digits }***.

- Function `echo(str1,str2,...,strN)` prints the concatenation of given string arguments on the console.
- Function `string(_#siz,str1,str2,...,strN)` generates a vector corresponding to the concatenation of given string/number arguments.

# Dynamic arrays:

A dynamic array is defined as a one-column (or empty) image `[ind]` in the image list. It allows elements to be added or removed, each element having the same dimension (which is actually the number of channels of image `[ind]`). Dynamic arrays adapt their size to the number of elements they contain.

A dynamic array can be manipulated in a math expression, with the following functions:

- `da_size(_#ind)`: Return the number of elements in dynamic array `[ind]`.
- `da_back(_#ind)`: Return the last element of the dynamic array `[ind]`.
- `da_insert(_#ind,pos,elt_1,_elt_2,...,_elt_N)`: Insert `N` new elements `elt_k` starting from index `pos` in dynamic array `[ind]`.
- `da_push(_#ind,elt1,_elt2,...,_eltN)`: Insert `N` new elements `elt_k` at the end of dynamic array `[ind]`.
- `da_pop(_#ind)`: Same as `da_back()` but also remove last element from the dynamic array `[ind]`.
- `da_push_heap(_#ind,elt1,_elt2,...,_eltN)` and `da_pop_heap(_#ind)` does the same but for a dynamic array viewed as a min-heap structure.
- `da_remove(_#ind,_start,_end)`: Remove elements located between indices `start` and `end` (included) in dynamic array `[ind]`.
- `da_freeze(_#ind)`: Convert a dynamic array into a 1-column image with height `da_size(#ind)`.
- The value of the k-th element of dynamic array `[ind]` is retrieved with `i[_#ind,k]` (if the element is a scalar value), or `I[_#ind,k]` (if the element is a vector).

In the functions above, argument `#ind` may be omitted in which case it is assumed to be `#-1`.

# Special operators:

- `;`: expression separator. The returned value is always the last encountered expression. For instance expression `1;2;pi` is evaluated as `pi`.
- `=`: variable assignment. Variables in mathematical parser can only refer to numerical values (vectors or scalars). Variable names are case-sensitive. Use this operator in conjunction with `;` to define more complex evaluable expressions, such as

```
t = cos(x); 3*t^2 + 2*t + 1
```

These variables remain **local** to the mathematical parser and cannot be accessed outside the evaluated expression.

- Variables defined in math parser may have a **constant** property, by specifying keyword `const` before the variable name (e.g. 'const foo = pi/4;'). The value set to such a variable must be indeed a **constant scalar**. Constant variables allows certain types of optimizations in the math JIT compiler.

## Specific functions:

- `addr(expr)` : return the pointer address to the specified expression `expr`.
- `o2c(_#ind,offset)` and `c2o(_#ind,x,_y,_z,_c)` : Convert image offset to image coordinates and vice-versa.
- `fill(target,expr)` or `fill(target,index_name,expr)` fill the content of the specified target (often vector-valued) using a given expression, e.g. `V = vector16(); fill(V,k,k^2 + k + 1);`. For a vector-valued target, it is basically equivalent to: `for (index_name = 0, index_name<size(target), ++index_name, target[index_name] = expr);`.
- `u(max)` or `u(min,max,_include_min,_include_max)` : return a random value in range `0...max` or `min...max`, following a uniform distribution. Each range extremum can be included (default) in the distribution or not.
- `v(max)` or `v(min,max,_include_min,_include_max)` do the same but returns an integer in specified range.
- `f2ui(value)` and `ui2f(value)` : Convert a large unsigned integer as a negative floating point value (and vice-versa), so that 32bits floats can be used to store large integers while keeping a unitary precision.
- `i(_a,_b,_c,_d,_interpolation_type,_boundary_conditions)` : return the value of the pixel located at position `(a,b,c,d)` in the associated image, if any (`0` otherwise). `interpolation_type` can be ***`{ 0:nearest neighbor | 1:linear | 2:cubic }`***. `boundary_conditions` can be ***`{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`***. Omitted coordinates are replaced by their default values which are respectively `x`, `y`, `z`, `c`, `interpolation` and `boundary`. For instance command

```
fill 0.5*(i(x+1)-i(x-1))
```

will estimate the X-derivative of an image with a classical finite difference scheme.

- `j(_dx,_dy,_dz,_dc,_interpolation_type,_boundary_conditions)` does the same for the pixel located at position `(x+dx,y+dy,z+dz,c+dc)` (pixel access relative to the current coordinates).
- `i[offset,_boundary_conditions]` returns the value of the pixel located at specified `offset` in the associated image buffer (or `0` if offset is out-of-bounds).
- `j[offset,_boundary_conditions]` does the same for an offset relative to the current pixel coordinates `(x,y,z,c)`.
- `i(#ind,_x,_y,_z,_c,_interpolation,_boundary_conditions)`, `j(#ind,_dx,_dy,_dz,_dc,_interpolation,_boundary_conditions)`, `i[#ind,offset,_boundary_conditions]` and `i[offset,_boundary_conditions]` are similar expressions used to access pixel values for any numbered image `[ind]` of the list.
- `I/J[_#ind,offset,_boundary_conditions]` and `I/J(_#ind,_x,_y,_z,_interpolation,_boundary_conditions)` do the same as `i/j[_#ind,offset,_boundary_conditions]` and `i/j(_#ind,_x,_y,_z,_c,_interpolation,_boundary_conditions)` but return a vector instead of a scalar (e.g. a vector `[ R,G,B ]` for a pixel at `(a,b,c)` in a color image).

- `crop(_#ind,_x,_y,_z,_c,_dx,_dy,_dz,_dc,_boundary_conditions)` returns a vector whose values come from the cropped region of image `[ind]` (or from default image selected if `ind` is not specified). Cropped region starts from point `(x,y,z,c)` and has a size of `(dx,dy,dz,dc)`. Arguments for coordinates and sizes can be omitted if they are not ambiguous (e.g. `crop(#ind,x,y,dx,dy)` is a valid invocation of this function).

* `crop(S,w,h,d,s,_x,_y,_z,_c,_dx,_dy,_dz,_dc,_boundary_conditions)` does the same but extracts the cropped data from a vector `S`, viewed as an image of size `(w,h,d,s)`.
- `draw(_#ind,S,_x,_y,_z,_c,_dx,_dy,_dz,_dc,_opacity,_opacity_mask,_max_opacity_m` draws a sprite `S` in image `[ind]` (or in default image selected if `ind` is not specified) at coordinates `(x,y,z,c)`.

- `draw(D,w,h,s,d,S,_x,_y,_z,_c,_dx,_dy,_dz,_dc,_opacity,_M,_max_M)` does the same but draw the sprite `S` in the vector `D`, viewed as an image of size `(w,h,d,s)`.

- `polygon(_#ind,nb_vertices,coords,_opacity,_color)` draws a filled polygon in image `[ind]` (or in default image selected if `ind` is not specified) at specified coordinates. It draws a single line if `nb_vertices` is set to 2.

- `polygon(_#ind,-nb_vertices,coords,_opacity,_pattern,_color)` draws a outlined polygon in image `[ind]` (or in default image selected if `ind` is not specified) at specified coordinates and with specified line pattern. It draws a single line if `nb_vertices` is set to 2.

- `ellipse(_#ind,xc,yc,radius1,_radius2,_angle,_opacity,_color)` draws a filled ellipse in image `[ind]` (or in default image selected if `ind` is not specified) with specified coordinates.

- `ellipse(_#ind,xc,yc,-radius1,-_radius2,_angle,_opacity,_pattern,_color)` draws an outlined ellipse in image `[ind]` (or in default image selected if `ind` is not specified).

- `flood(_#ind,_x,_y,_z,_tolerance,_is_high_connectivity,_opacity,_color)` performs a flood fill in image `[ind]` (or in default image selected if `ind` is not specified) with specified coordinates. This is the math evaluator analog to command **flood**.

- `resize(#ind,w,_h,_d,_s,_interp,_boundary_conditions,_cx,_cy,_cz,_cc)` resizes an image of the associated list with specified dimension and interpolation method. When using this function, you should consider retrieving the (non-constant) image dimensions using the dynamic functions `w(_#ind)`, `h(_#ind)`, `d(_#ind)`, `s(_#ind)`, `wh(_#ind)`, `whd(_#ind)` and `whds(_#ind)` instead of the corresponding constant variables.

- `if(condition,expr_then,_expr_else)`: return value of `expr_then` or `expr_else`, depending on the value of `condition` ***`{ 0:false | other:true }`***. `expr_else` can be omitted in which case `0` is returned if the condition does not hold. Using the ternary operator `condition?expr_then[:expr_else]` gives an equivalent expression. For instance, G'MIC commands

```
fill if(!(x%10),255,i)
```

and

```
fill x%10?i:255
```

both draw blank vertical lines on every 10th column of an image.
- `do(expression,_condition)` repeats the evaluation of `expression` until `condition` vanishes (or until `expression` vanishes if no `condition` is specified). For instance, the expression:

```
if(N<2,N,n=N-1;F0=0;F1=1;do(F2=F0+F1;F0=F1;F1=F2,n=n-1))
```

returns the N-th value of the Fibonacci sequence, for `N>=0` (e.g., `46368` for `N=24`). `do(expression,condition)` always evaluates the specified expression at least once, then check for the loop condition. When done, it returns the last value of `expression`.

- `for(init,condition,_procedure,body)` first evaluates the expression `init`, then iteratively evaluates `body` (followed by `procedure` if specified) while `condition` holds (i.e. not zero). It may happen that no iterations are done, in which case the function returns `nan`. Otherwise, it returns the last value of `body`. For instance, the expression:

```
if(N<2,N,for(n=N;F0=0;F1=1,n=n-1,F2=F0+F1;F0=F1;F1=F2))
```

returns the `N`-th value of the Fibonacci sequence, for `N>=0` (e.g., `46368` for `N=24`).

- `while(condition,expression)` is exactly the same as `for(init,condition,expression)` without the specification of an initializing expression.
- `repeat(nb_iters,expr)` or `fill(nb_iters,iter_name,expr)` run `nb_iters` iterations of the specified expression `expr`, e.g. `V = vector16(); repeat(16,k,V[k] = k^2 + k + 1);`. It is basically equivalent to: `for (iter_name = 0, iter_name<nb_iters, ++iter_name, expr);`.
- `break()` and `continue()` respectively breaks and continues the current running block.
- `fsize('filename`)' returns the size of the specified `filename` (or `-1` if file does not exist).
- `date(attr,'path`)' returns the date attribute for the given `path` (file or directory), with `attr` being ***{ 0:year | 1:month | 2:day | 3:day of week | 4:hour | 5:minute | 6:second }***, or a vector of those values.
- `date(_attr)` returns the specified attribute for the current (locale) date (attributes being ***{ 0...6:same meaning as above | 7:milliseconds }***).
- `print(expr1,expr2,...)` or `print(#ind)` prints the value of the specified expressions (or image information) on the console, and returns the value of the last expression (or `nan` in case of an image). Function `prints(expr)` also prints the string composed of the character codes defined by the vector-valued expression (e.g. `prints('Hello`)').
- `debug(expression)` prints detailed debug info about the sequence of operations done by the math parser to evaluate the expression (and returns its value).
- `display(_X,_w,_h,_d,_s)` or `display(#ind)` display the contents of the vector `X` (or specified image) and wait for user events. if no arguments are provided, a memory snapshot of the math parser environment is displayed instead.
- `begin(expression)` and `end(expression)` evaluates the specified expressions only once, respectively at the beginning and end of the evaluation procedure, and this, even when multiple evaluations are required (e.g. in 'fill ">begin(foo = 0); ++foo"').
- `copy(dest,src,_nb_elts,_inc_d,_inc_s,_opacity)` copies an entire memory block of `nb_elts` elements starting from a source value `src` to a specified destination `dest`, with increments defined by `inc_d` and `inc_s` respectively for the destination and source pointers.
- `stats(_#ind)` returns the statistics vector of the running image `[ind]`, i.e the vector `[ im,iM,ia,iv,xm,ym,zm,cm,xM,yM,zM,cM,is,ip ]` (14 values).
- `ref(expr,a)` references specified expression `expr` as variable name `a`.
- `unref(a,b,...)` destroys references to the named variable given as arguments.

- `breakpoint()` inserts a possible computation breakpoint (useless with the cli interface).
- '_(comment) expr' just returns expression `expr` (useful for inserting inline comments in math expressions).
- `run('pipeline`)' executes the specified G'MIC pipeline as if it was called outside the currently evaluated expression.
- `set('variable_name',A)` set the G'MIC variable `$variable_name` with the value of expression `A`. If `A` is a vector-valued variable, it is assumed to encode a string.
- `store('variable_name',A,_w,_h,_d,_s,_is_compressed)` transfers the data of vector `A` as a `(w,h,d,s)` image to the G'MIC variable `$variable_name`. Thus, the data becomes available outside the math expression (that is equivalent to using the regular command **store**, but directly in the math expression).
- `get('variable_name',_size,_return_as_string)` returns the value of the specified variable, as a vector of `size` values, or as a scalar (if `size` is zero or not specified).
- `name(_#ind,size)` returns a vector of size `size`, whose values are the characters codes of the name of image `[ind]` (or default image selected if `ind` is not specified).
- `correlate(I,wI,hI,dI,sI,K,wK,hK,dK,sK,_boundary_conditions,_is_normalized,_cha` returns the correlation, unrolled as a vector, of the `(wI,hI,dI,sI)`-sized image `I` with the `(wK,hK,dK,sK)`-sized kernel `K` (the meaning of the other arguments are the same as in command `correlate`). Similar function `convolve(...)` is also defined for computing the convolution between `I` and `K`.

# User-defined macros:

- Custom macro functions can be defined in a math expression, using the assignment operator `=`, e.g.

  ```
  foo(x,y) = cos(x + y); result = foo(1,2) + foo(2,3)
  ```

- Trying to override a built-in function (e.g. `abs()`) has no effect.
- Overloading macros with different number of arguments is possible. Re-defining a previously defined macro with the same number of arguments discards its previous definition.
- Macro functions are indeed processed as **macros** by the mathematical evaluator. You should avoid invoking them with arguments that are themselves results of assignments or self-operations. For instance,

  ```
  foo(x) = x + x; z = 0; foo(++z)
  ```

returns `4` rather than expected value `2`.

- When substituted, macro arguments are placed inside parentheses, except if a number sign `#` is located just before or after the argument name. For instance, expression

  ```
  foo(x,y) = x*y; foo(1+2,3)
  ```

returns `9` (being substituted as `(1+2)*(3)`), while expression

```
foo(x,y) = x#*y#; foo(1+2,3)
```

returns `7` (being substituted as `1+2*3`).

- Number signs appearing between macro arguments function actually count for **empty** separators. They may be used to force the substitution of macro arguments in unusual places, e.g. as in

  > ```
  > str(N) = ['I like N#'];
  > ```

- Macros with variadic arguments can be defined, by specifying a single argument name followed by `...`. For instance,

  > ```
  > foo(args...) = sum([ args ]^2);
  > ```

defines a macro that returns the sum of its squared arguments, so `foo(1,2,3)` returns `14` and `foo(4,5)` returns `41`.

## Multi-threaded and in-place evaluation:

- If your image data are large enough and you have several CPUs available, it is likely that the math expression passed to a `fill`, `eval` or `input` commands is evaluated in parallel, using multiple computation threads.
- Starting an expression with `:` or `*` forces the evaluations required for an image to be run in parallel, even if the amount of data to process is small (beware, it may be slower to evaluate in this case!). Specify `:` (rather than `*`) to avoid possible image copy done before evaluating the expression (this saves memory, but do this only if you are sure this step is not required!)
- Expression starting with `+` are evaluated in a single-threaded way, with possible image copy.
- If the specified expression starts with `>` or `<`, the pixel access operators `i()`, `i[]`, `j()` and `j[]` return values of the image being currently modified, in forward (`>`) or backward (`<`) order. The multi-threading evaluation of the expression is disabled in this case.
- Function `critical(expr)` forces the execution of the given expression in a single thread at a time.
- `begin_t(expr)` and `end_t(expr)` evaluates the specified expression once for each running thread (so possibly several times) at the beginning and the end of the evaluation procedure.
- `merge(variable,operator)` tells to merge the local variable value computed by threads, with the specified operator, when all threads have finished computing.
- Expressions `i(_#ind,x,_y,_z,_c)=value`, `j(_#ind,x,_y,_z,_c)=value`, `i[_#ind,offset]=value` and `j[_#ind,offset]=value` set a pixel value at a different location than the running one in the image `[ind]` (or in the associated image if argument `#ind` is omitted), either with global coordinates/offsets (with `i(...)` and `i[...]`), or relatively to the current position `(x,y,z,c)` (with `j(...)` and `j[...]`). These expressions always return `value`.

# Adding Custom Commands

- New custom commands can be added by the user, through the use of **G'MIC custom commands files**.
- A command file is a simple text file, where each line starts either by

```
command_name: command_definition
```

or

```
command_definition (continuation)
```

- At startup, G'MIC automatically includes user's command file `$HOME/.gmic` (on **Unix**) or `%USERPROFILE%\user.gmic` (on **Windows**). The CLI tool `gmic` automatically runs the command `cli_start` if defined.
- Custom command names must use character set `[a-zA-Z0-9_]` and cannot start with a number.
- Any `# comment` expression found in a custom commands file is discarded by the G'MIC parser, wherever it is located in a line.
- In a custom command, the following `$-expressions` are recognized and substituted:
  - `$*` is substituted by a verbatim copy of the specified string of arguments (do not include arguments set to default values).
  - `$"*"` is substituted by the sequence of specified arguments, separated by commas `,`, each being double-quoted (include arguments set to default values).
  - `$#` is substituted by the maximum index of known arguments (either specified by the user or set to a default value in the custom command).
  - `$[]` is substituted by the list of selected image indices that have been specified in the command invocation.
  - `$?` is substituted by a printable version of `$[]` to be used in command descriptions.
  - `$i` and `${i}` are both substituted by the `i`-th specified argument. Negative indices such as `${-j}` are allowed and refer to the `j`-th latest argument. `$0` is substituted by the custom command name.
  - `${i=default}` is substituted by the value of `$i` (if defined) or by its new value set to `default` otherwise (`default` may be a `$-expression` as well).
  - `${subset}` is substituted by the argument values (separated by commas `,`) of a specified argument subset. For instance expression `${2--2}` is substituted by all specified command arguments except the first and the last one. Expression `${^0}` is then substituted by all arguments of the invoked command (eq. to `$*` if all arguments have been indeed specified).
  - `$=var` is substituted by the set of instructions that will assign each argument `$i` to the named variable `var$i` (for i in `[0...$#]`. This is particularly useful when a custom command want to manage variable numbers of arguments. Variables names must use character set `[a-zA-Z0-9_]` and cannot start with a number.

- These particular `$-expressions` for custom commands are **always substituted**, even in double-quoted items or when the dollar sign `$` is escaped with a backslash `$`. To avoid substitution, place an empty double quoted string just after the `$` (as in `$""1`).
- Specifying arguments may be skipped when invoking a custom command, by replacing them by commas `,` as in expression

```
flower ,,3
```

Omitted arguments are set to their default values, which must be thus explicitly defined in the code of the corresponding custom command (using default argument expressions as `${1=default}`).

- If one numbered argument required by a custom command misses a value, an error is thrown by the G'MIC interpreter.
- It is possible to specialize the invocation of a `+command` by defining it as

  ```
  +command_name: command_definition
  ```

- A +-specialization takes priority over the regular command definition when the command is invoked with a prepended `+`.
- When only a +-specialization of a command is defined, invoking `command` is actually equivalent to `+command`.

# List of Commands

All available **G'MIC** commands are listed below, by categories. An argument specified between `[]` or starting by `_` is optional except when standing for an existing image `[image]`, where `image` can be either an index number or an image name. In this case, the `[]` characters are mandatory when writing the item. Note that all images that serve as illustrations in this reference documentation are normalized in range `[0,255]` before being displayed. You may need to do this explicitly (command `normalize 0,255`) if you want to save and view images with the same aspect than those illustrated in the example codes.
The examples accompanying this 'List of Commands' illustrate the use of the **G'MIC** language and are written as they would appear in a custom command. While some examples may work if entered directly at a shell prompt, there is no guarantee. No attempt has been made to escape special characters in these examples, which many shells reserve.

# Categories:

- **Other Interactive Commands**
- **Command Shortcuts**

# Global Options:

| | | |
|---|---|---|
| `debug` | help | version |

# Input / Output:

| | | | | |
|---|---|---|---|---|
| `camera` | `command` | compress_to_keypoints | `cursor` | `delete` |
| display | display0 | display_array | display_camera | display_clut |
| display_fft | display_graph | display_histogram | display_parametric | display_polar |
| display_quiver | display_rgba | display_tensors | display_voxels3d | display_warp |
| `echo` | echo_file | font | font2gmz | function1d |
| identity | `input` | input_565 | input_bytes | input_csv |
| input_cube | input_flo | input_glob | input_gpl | input_cached |
| input_obj | input_text | lorem | `network` | `output` |
| output_565 | output_cube | output_flo | output_ggr | output_gmz |
| output_obj | output_text | outputn | outputp | outputw |
| outputx | parse_cli | parse_gmd | gmd2html | gmd2ascii |
| parse_gui | `pass` | `plot` | poincare_disk | portrait |
| print | random_pattern | `screen` | `select` | `serialize` |
| shape_circle | shape_cupid | shape_diamond | shape_dragon | shape_fern |
| shape_gear | shape_heart | shape_menger | shape_mosely | shape_polygon |
| shape_rays | shape_snowflake | shape_star | `shared` | sample |
| `srand` | `store` | testimage2d | `uncommand` | uniform_distribution |
| `unserialize` | update | `verbose` | `wait` | `warn` |
| `window` | | | | |

# List Manipulation:

| | | | | |
|---|---|---|---|---|
| `keep` | keep_named | `move` | `name` | `remove` |
| remove_duplicates | remove_empty | remove_named | `reverse` | sort_list |

# Mathematical Operators:

| | | | | |
|---|---|---|---|---|
| `abs` | `acos` | `acosh` | `add` | `and` |
| argmax | argmaxabs | argmin | argminabs | `asin` |
| `asinh` | `atan` | `atan2` | `atanh` | `bsl` |
| `bsr` | `cos` | `cosh` | deg2rad | `div` |
| div_complex | `eq` | `erf` | `exp` | `ge` |
| `gt` | `le` | `lt` | `log` | `log10` |
| `log2` | `max` | `maxabs` | `mdiv` | med |
| `min` | `minabs` | `mod` | `mmul` | `mul` |
| mul_channels | mul_complex | `neq` | `or` | `pow` |
| rad2deg | `rol` | `ror` | `sign` | `sin` |
| `sinc` | `sinh` | `sqr` | `sqrt` | `sub` |
| `tan` | `tanh` | `xor` | | |

# Values Manipulation:

| | | | | |
|---|---|---|---|---|
| apply_curve | apply_gamma | balance_gamma | cast | complex2polar |
| compress_clut | compress_huffman | huffman_tree | compress_rle | `cumulate` |
| `cut` | decompress_clut | decompress_from_keypoints | decompress_huffman | decompress_rle |
| `discard` | eigen2tensor | `endian` | `equalize` | `fill` |
| `index` | inrange | `map` | mix_channels | negate |
| `noise` | noise_perlin | noise_poissondisk | normp | norm1 |
| norm2 | `normalize` | normalize_l2 | normalize_sum | not |
| orientation | oneminus | otsu | polar2complex | quantize |
| quantize_area | `rand` | rand_sum | replace | replace_inf |
| replace_infnan | replace_nan | replace_seq | replace_str | `round` |
| roundify | `set` | threshold | vector2tensor | |

# Colors:

| | | | | |
|---|---|---|---|---|
| adjust_colors | apply_channels | autoindex | bayer2rgb | clut |

|  |  |  |  |  |
|---|---|---|---|---|
| clut2hald | hald2clut | cmy2rgb | cmyk2rgb | colorblind |
| colormap | compose_channels | count_colors | deltaE | direction2rgb |
| ditheredbw | fill_color | gradient2rgb | hcy2rgb | hsi2rgb |
| hsi82rgb | hsl2rgb | hsl82rgb | hsv2rgb | hsv82rgb |
| int2rgb | ipremula | jzazbz2rgb | jzazbz2xyz | lab2lch |
| lab2rgb | lab2srgb | lab82srgb | lab2xyz | lab82rgb |
| lch2lab | lch2rgb | lch82rgb | luminance | lightness |
| lut_contrast | map_clut | match_histogram | match_icp | match_pca |
| match_rgb | mix_rgb | oklab2rgb | palette | premula |
| pseudogray | random_clut | random_clut | replace_color | retinex |
| rgb2bayer | rgb2cmy | rgb2cmyk | rgb2hcy | rgb2hsi |
| rgb2hsi8 | rgb2hsl | rgb2hsl8 | rgb2hsv | rgb2hsv8 |
| rgb2int | rgb2jzazbz | rgb2lab | rgb2lab8 | rgb2lch |
| rgb2lch8 | rgb2luv | rgb2oklab | rgb2ryb | rgb2srgb |
| rgb2xyz | rgb2xyz8 | rgb2yiq | rgb2yiq8 | rgb2ycbcr |
| rgb2yuv | rgb2yuv8 | remove_opacity | ryb2rgb | select_color |
| sepia | solarize | split_colors | split_opacity | split_vector |
| srgb2lab | srgb2lab8 | srgb2rgb | to_a | to_color |
| to_colormode | to_gray | to_graya | to_pseudogray | to_rgb |
| to_rgba | to_automode | xyz2jzazbz | xyz2lab | xyz2rgb |
| xyz82rgb | ycbcr2rgb | yiq2rgb | yiq82rgb | yuv2rgb |
| yuv82rgb |  |  |  |  |

# Geometry Manipulation:

|  |  |  |  |  |
|---|---|---|---|---|
| `append` | append_tiles | apply_scales | `autocrop` | autocrop_components |
| autocrop_seq | channels | columns | `crop` | diagonal |
| edgels | elevate | expand_x | expand_xy | expand_xyz |
| expand_y | expand_z | extract | extract_region | montage |
| `mirror` | `permute` | rescale2d | rescale3d | `resize` |
| resize_as_image | resize_mn | resize_pow2 | `rotate` | rotate_tileable |
| rows | scale2x | scale3x | scale_dcci2x | seamcarve |
| `shift` | shrink_x | shrink_xy | shrink_xyz | shrink_y |
| shrink_z | slices | `sort` | `split` | split_tiles |
| undistort | `unroll` | upscale_smart | volumetric2d |  |

# Filtering:

| | | | | |
|---|---|---|---|---|
| bandpass | `bilateral` | `blur` | blur_angular | blur_bloom |
| blur_linear | blur_radial | blur_selective | blur_x | blur_xy |
| blur_xyz | blur_y | blur_z | `boxfilter` | bump2normal |
| closing | closing_circ | compose_freq | `convolve` | convolve_fft |
| `correlate` | cross_correlation | curvature | dct | deblur |
| deblur_goldmeinel | deblur_richardsonlucy | deconvolve_fft | deinterlace | `denoise` |
| denoise_haar | denoise_cnn | denoise_patchpca | `deriche` | `dilate` |
| dilate_circ | dilate_oct | dilate_threshold | divergence | dog |
| diffusiontensors | edges | `erode` | erode_circ | erode_oct |
| erode_threshold | `fft` | gradient | gradient_norm | gradient_orientation |
| `guided` | haar | heat_flow | hessian | idct |
| iee | `ifft` | ihaar | ilaplacian | inn |
| `inpaint` | inpaint_pde | inpaint_flow | inpaint_holes | inpaint_morpho |
| inpaint_matchpatch | kuwahara | laplacian | lic | map_tones |
| map_tones_fast | meancurvature_flow | `median` | merge_alpha | nlmeans |
| nlmeans_core | normalize_local | normalized_cross_correlation | opening | opening_circ |
| percentile | peronamalik_flow | phase_correlation | pde_flow | periodize_poisson |
| rbf | red_eye | remove_hotpixels | remove_pixels | rolling_guidance |
| sharpen | sharpen_alpha | `smooth` | split_freq | solve_poisson |
| split_alpha | split_details | structuretensors | solidify | syntexturize |
| syntexturize_matchpatch | tv_flow | unsharp | unsharp_octave | `vanvliet` |
| voronoi | watermark_fourier | `watershed` | | |

# Features Extraction:

| | | | | |
|---|---|---|---|---|
| area | area_fg | at_line | at_quadrangle | barycenter |
| betti | canny | delaunay | detect_skin | `displacement` |

| | | | | |
|---|---|---|---|---|
| `distance` | fftpolar | `histogram` | histogram_masked | histogram_nd |
| histogram_cumul | histogram_pointwise | hough | ifftpolar | img2patches |
| isophotes | `label` | label_fg | laar | max_patch |
| min_patch | minimal_path | mse | mse_matrix | patches2img |
| patches | `matchpatch` | plot2value | pointcloud | psnr |
| psnr_matrix | segment_watershed | shape2bump | skeleton | slic |
| ssd_patch | ssim | ssim_matrix | thinning | tones |
| topographic_map | tsp | variance_patch | | |

## Image Drawing:

| | | | | |
|---|---|---|---|---|
| arrow | axes | ball | chessboard | cie1931 |
| circle | close_binary | curve | `ellipse` | `flood` |
| gaussian | `graph` | grid | `image` | imagealpha |
| `line` | line_aa | spline | thickline | thickspline |
| `mandelbrot` | marble | maze | maze_mask | newton_fractal |
| `object3d` | pack_sprites | piechart | `plasma` | `point` |
| polka_dots | `polygon` | quiver | rectangle | rorschach |
| sierpinski | spiralbw | tetraedron_shade | `text` | text_outline |
| triangle_shade | truchet | turbulence | yinyang | |

## Matrix Computation:

| | | | | |
|---|---|---|---|---|
| dijkstra | `eigen` | eye | fitsamples | `invert` |
| meigen | `mproj` | orthogonalize | poweriteration | `solve` |
| `svd` | transpose | `trisolve` | | |

## 3D Meshes:

| | | | | |
|---|---|---|---|---|
| `add3d` | animate3d | apply_camera3d | apply_matrix3d | array3d |
| arrow3d | axes3d | boundingbox3d | box3d | center3d |
| chainring3d | circle3d | circles3d | color3d | colorcube3d |
| colorize3d | cone3d | cubes3d | cup3d | curve3d |

| | | | | |
|---|---|---|---|---|
| cylinder3d | delaunay3d | distribution3d | `div3d` | double3d |
| elevation3d | empty3d | extract_textures3d | extrude3d | focale3d |
| fov3d | gaussians3d | gmic3d | gyroid3d | histogram3d |
| image6cube3d | imageblocks3d | imagecube3d | imageplane3d | imagepyramid3d |
| imagerubik3d | imagesphere3d | `isoline3d` | `isosurface3d` | label3d |
| label_points3d | lathe3d | `light3d` | line3d | lines3d |
| lissajous3d | mode3d | moded3d | `mul3d` | normalize3d |
| opacity3d | parametric3d | pca_patch3d | plane3d | point3d |
| pointcloud3d | pose3d | primitives3d | projections3d | pyramid3d |
| quadrangle3d | random3d | reverse3d | `rotate3d` | rotation3d |
| sierpinski3d | size3d | skeleton3d | snapshot3d | specl3d |
| specs3d | sphere3d | spherical3d | spline3d | split3d |
| sprite3d | sprites3d | star3d | `streamline3d` | `sub3d` |
| subdivide3d | superformula3d | surfels3d | tensors3d | text_pointcloud3d |
| text3d | texturize3d | torus3d | triangle3d | volume3d |
| voxelize3d | weird3d | | | |

## Flow Control:

| | | | | |
|---|---|---|---|---|
| apply_parallel | apply_parallel_channels | apply_parallel_overlap | apply_tiles | apply_timeout |
| `check` | `check3d` | `continue` | `break` | `do` |
| `done` | `elif` | `else` | `fi` | `error` |
| `eval` | `exec` | exec_out | `for` | `foreach` |
| `if` | `local` | `mutex` | `noarg` | `onfail` |
| `parallel` | `progress` | `quit` | `repeat` | `return` |
| rprogress | run | `skip` | `status` | `while` |

## Neural Networks:

| | | | | |
|---|---|---|---|---|
| nn_lib | nn_init | nn_check_layer | nn_add | nn_append |
| nn_avgpool2d | nn_avgpool3d | nn_clone | nn_conv2d | nn_conv2dnl |
| nn_conv2dnnl | nn_conv3d | nn_conv3dnl | nn_conv3dnnl | nn_crop |
| nn_distance | nn_dropout | nn_fc | nn_nlfc | nn_fcnl |
| nn_fcnnl | nn_input | nn_maxpool2d | nn_maxpool3d | nn_mul |

| | | | | |
|---|---|---|---|---|
| nn_nl | nn_normalize | nn_patchdown2d | nn_patchdown3d | nn_patchup2d |
| nn_patchup3d | nn_rename | nn_resconv2d | nn_resconv2dnl | nn_resconv2dnnl |
| nn_resconv3d | nn_resconv3dnl | nn_resconv3dnnl | nn_resfc | nn_resfcnl |
| nn_resfcnnl | nn_reshape | nn_resize | nn_run | nn_split |
| nn_loss_binary_crossentropy | nn_loss_crossentropy | nn_loss_mse | nn_loss_normp | nn_loss_softmax_crossentropy |
| nn_print | nn_trainer | nn_size | nn_load | nn_save |
| nn_store | | | | |

# Arrays, Tiles and Frames:

| | | | | |
|---|---|---|---|---|
| array | array_fade | array_mirror | array_random | frame_blur |
| frame_cube | frame_fuzzy | frame_painting | frame_pattern | frame_round |
| frame_seamless | frame_x | frame_xy | frame_xyz | frame_y |
| img2ascii | imagegrid | imagegrid_hexagonal | imagegrid_triangular | linearize_tiles |
| map_sprites | pack | puzzle | quadratize_tiles | rotate_tiles |
| shift_tiles | taquin | tunnel | | |

# Artistic:

| | | | | |
|---|---|---|---|---|
| boxfitting | brushify | cartoon | color_ellipses | cubism |
| draw_whirl | drawing | drop_shadow | drop_shadow | ellipsionism |
| fire_edges | fractalize | glow | halftone | hardsketchbw |
| hearts | houghsketchbw | lightrays | light_relief | linify |
| mosaic | old_photo | pencilbw | pixelsort | polaroid |
| polygonize | poster_edges | poster_hope | rodilius | sketchbw |
| sponge | stained_glass | stars | stencil | stencilbw |
| stylize | tetris | warhol | weave | whirls |

# Warpings:

| | | | | |
|---|---|---|---|---|
| deform | euclidean2polar | equirectangular2nadirzenith | fisheye | flower |

| | | | | |
|---|---|---|---|---|
| kaleidoscope | map_sphere | nadirzenith2equirectangular | polar2euclidean | raindrops |
| ripple | rotoidoscope | spherize | symmetrize | transform_polar |
| twirl | `warp` | warp_patch | warp_perspective | warp_rbf |
| water | wave | wind | zoom | |

# Degradations:

| | | | | |
|---|---|---|---|---|
| cracks | light_patch | noise_hurl | pixelize | scanlines |
| shade_stripes | shadow_patch | spread | stripes_y | texturize_canvas |
| texturize_paper | vignette | watermark_visible | | |

# Blending and Fading:

| | | | | |
|---|---|---|---|---|
| blend | blend_edges | blend_fade | blend_median | blend_seamless |
| fade_diamond | fade_linear | fade_radial | fade_x | fade_y |
| fade_z | sub_alpha | | | |

# Image Sequences and Videos:

| | | | | |
|---|---|---|---|---|
| animate | apply_camera | apply_files | apply_video | average_files |
| average_video | fade_files | fade_video | files2video | median_files |
| median_video | morph | morph_files | morph_rbf | morph_video |
| register_nonrigid | register_rigid | transition | transition3d | video2files |

# Convenience Functions:

| | | | | |
|---|---|---|---|---|
| add_copymark | alert | arg | arg0 | arg2img |
| arg2var | autocrop_coords | average_vectors | base642img | base642uint8 |
| basename | bin | bin2dec | cat | color2name |
| covariance_vectors | da_freeze | dec | dec2str | dec2bin |
| dec2hex | dec2oct | fibonacci | file_mv | filename |

| | | | | |
|---|---|---|---|---|
| filename_rand | filename_dated | `files` | files2img | fitratio_wh |
| fitscreen | fontchart | fps | hex | hex2dec |
| hex2img | hex2str | img2base64 | img2hex | img2str |
| img2text | is_mesh3d | is_change | is_half | is_ext |
| is_image_arg | is_pattern | is_varname | is_videofilename | is_macos |
| is_windows | lof | math_lib | mad | max_w |
| max_h | max_d | max_s | max_wh | max_whd |
| max_whds | median_vectors | min_w | min_h | min_d |
| min_s | min_wh | min_whd | min_whds | name2color |
| `named` | narg | normalize_filename | oct | oct2dec |
| padint | path_cache | path_current | path_gimp | path_tmp |
| remove_copymark | reset | rgb | rgba | shell_cols |
| size_value | std_noise | str | strbuffer | str2hex |
| strcapitalize | strcontains | strclut | strlen | strreplace |
| strlowercase | struppercase | strvar | strcasevar | strver |
| tic | toc | uint82base64 | | |

# Other Interactive Commands:

| | | | | |
|---|---|---|---|---|
| demos | tixy | x_2048 | x_blobs | x_bouncing |
| x_color_curves | x_colorize | x_connect4 | x_crop | x_cut |
| x_fire | x_fireworks | x_fisheye | x_fourier | x_grab_color |
| x_hanoi | x_histogram | x_hough | x_jawbreaker | x_landscape |
| x_life | x_light | x_mandelbrot | x_mask_color | x_metaballs3d |
| x_minesweeper | x_minimal_path | x_morph | x_pacman | x_paint |
| x_plasma | x_quantize_rgb | x_reflection3d | x_rubber3d | x_segment |
| x_select_color | x_select_function1d | x_select_palette | x_shadebobs | x_spline |
| x_starfield3d | x_tetris | x_threshold | x_tictactoe | x_warp |
| x_waves | x_whirl | | | |

# Command Shortcuts:

| **Shortcut name** | **Equivalent command name** |
|---|---|
| **h** | help |
| **m** | `command` |

| | |
|---|---|
| **d** | display |
| **d0** | display0 |
| **da** | display_array |
| **dc** | display_camera |
| **dclut** | display_clut |
| **dfft** | display_fft |
| **dg** | display_graph |
| **dh** | display_histogram |
| **dq** | display_quiver |
| **drgba** | display_rgba |
| **dt** | display_tensors |
| **dv3d** | display_voxels3d |
| **dw** | display_warp |
| **e** | `echo` |
| **i** | `input` |
| **ib** | input_bytes |
| **ig** | input_glob |
| **it** | input_text |
| **o** | `output` |
| **ot** | output_text |
| **on** | outputn |
| **op** | outputp |
| **ow** | outputw |
| **ox** | outputx |
| **p** | print |
| **sh** | `shared` |
| **sp** | sample |
| **um** | uncommand |
| **up** | update |
| **v** | `verbose` |
| **w** | `window` |
| **k** | `keep` |
| **kn** | keep_named |
| **mv** | `move` |
| **nm** | `name` |
| **=>** | `name` |
| **rm** | `remove` |
| **rmn** | remove_named |
| **rv** | `reverse` |
| **+** | `add` |
| **&** | `and` |
| **<<** | `bsl` |
| **>>** | `bsr` |

| | |
|---|---|
| **/** | `div` |
| **==** | `eq` |
| **>=** | `ge` |
| **>** | `gt` |
| **<=** | `le` |
| **<** | `lt` |
| **m/** | `mdiv` |
| **%** | `mod` |
| **m*** | `mmul` |
| ***** | `mul` |
| **!=** | `neq` |
| **\|** | `or` |
| **^** | `pow` |
| **-** | `sub` |
| **c** | `cut` |
| **f** | `fill` |
| **ir** | inrange |
| **norm** | norm2 |
| **n** | `normalize` |
| **=** | `set` |
| **ac** | apply_channels |
| **fc** | fill_color |
| **a** | `append` |
| **z** | `crop` |
| **rs** | rescale2d |
| **rs3d** | rescale3d |
| **r** | `resize` |
| **ri** | resize_as_image |
| **s** | `split` |
| **y** | `unroll` |
| **b** | `blur` |
| **g** | `gradient` |
| **j** | `image` |
| **ja** | imagealpha |
| **j3d** | `object3d` |
| **t** | `text` |
| **to** | text_outline |
| **+3d** | `add3d` |
| **c3d** | center3d |
| **col3d** | color3d |
| **/3d** | `div3d` |
| **db3d** | double3d |
| **f3d** | focale3d |

| | |
|---|---|
| **l3d** | `light3d` |
| **m3d** | mode3d |
| **md3d** | moded3d |
| ***3d** | `mul3d` |
| **n3d** | normalize3d |
| **o3d** | opacity3d |
| **p3d** | primitives3d |
| **rv3d** | reverse3d |
| **r3d** | `rotate3d` |
| **sl3d** | specl3d |
| **ss3d** | specs3d |
| **s3d** | split3d |
| **-3d** | `sub3d` |
| **t3d** | texturize3d |
| **ap** | apply_parallel |
| **apc** | apply_parallel_channels |
| **apo** | apply_parallel_overlap |
| **at** | apply_tiles |
| **x** | `exec` |
| **xo** | exec_out |
| **l** | `local` |
| **q** | `quit` |
| **u** | `status` |
| **frame** | frame_xy |
| **nmd** | `named` |
| **xz** | x_crop |

# Examples of Use

`gmic` is a generic image processing tool which can be used in a wide variety of situations. The few examples below illustrate possible uses of this tool:

## View a list of images:

```
$ gmic file1.bmp file2.jpeg
```

## Convert an image file:

```
$ gmic input.bmp output output.jpg
```

## Create a volumetric image from a movie sequence:

```
$ gmic input.mpg append z output output.hdr
```

## Compute image gradient norm:

```
$ gmic input.bmp gradient_norm
```

## Denoise a color image:

```
$ gmic image.jpg denoise 30,10 output denoised.jpg
```

## Compose two images using overlay layer blending:

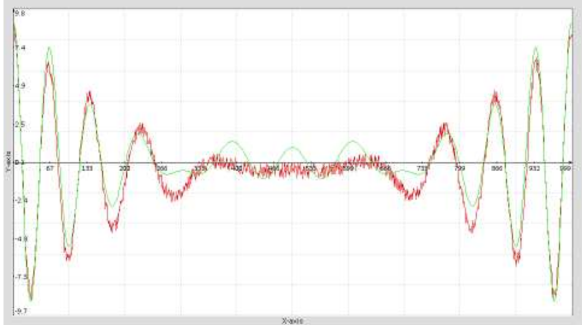
```
$ gmic image1.jpg image2.jpg blend overlay output blended.jpg
```

## Evaluate a mathematical expression:

```
$ gmic echo "cos(pi/4)^2+sin(pi/4)^2={cos(pi/4)^2+sin(pi/4)^2}"
```

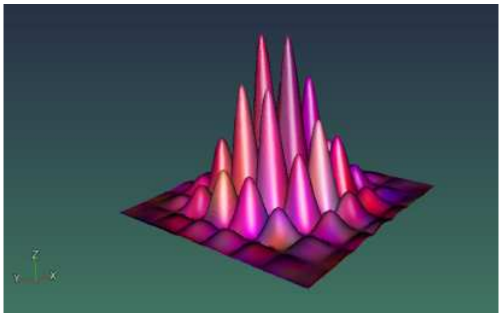
## Plot a 2D function:

```
$ gmic 1000,1,1,2 fill "X=3*(x-500)/500;X^2*sin(3*X^2)+(!c?u(0,
-1):cos(X*10))" plot
```



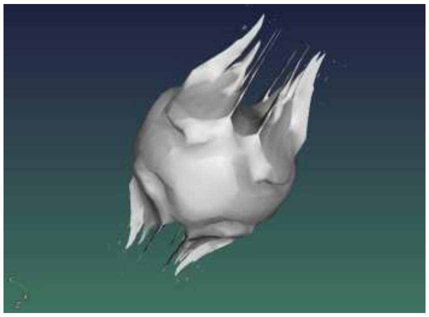
## Plot a 3D elevated function in random colors:

```
$ gmic 128,128,1,3,"u(0,255)
" plasma 10,3 blur 4 sharpen 10000 n 0,255 elevation3d[-1] "'X=(x-
64)/6;Y=(y-64)/6;100*exp(-(X^2+Y^2)/30)*abs(cos(X)*sin(Y))'"
```



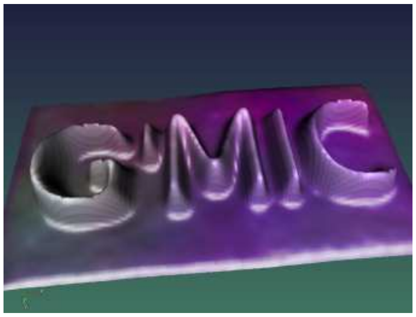
## Plot the isosurface of a 3D volume:

```
$ gmic mode3d 5 moded3d 5 double3d 0 isosurface3d "'x^2+y^2+abs(z)^abs(4*cos
3
```



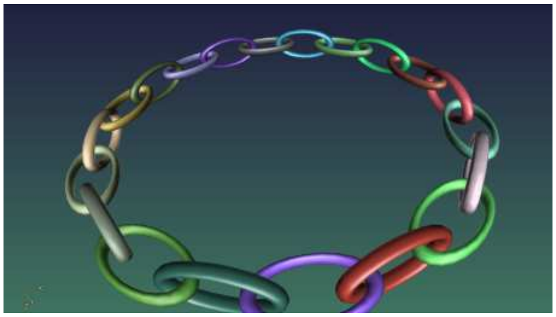
## Render a G'MIC 3D logo:

```
$ gmic 0 text G\'MIC,
0,0,53,1,1,1,1 expand_xy 10,0 blur 1 normalize 0,100 +plasma 0.4 add blur 1
```



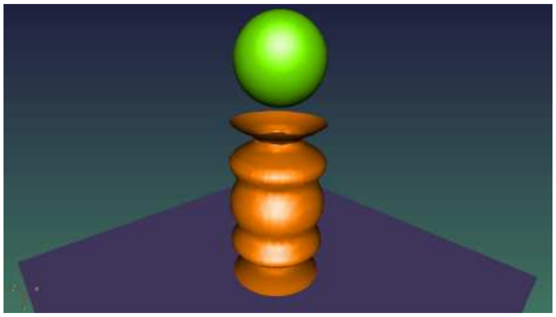
## Generate a 3D ring of torii:

```
$ gmic repeat 20 torus3d 15,2 color3d[-1] "{u(60,255)},{u(60,255)},
{u(60,255)}" *3d[-1] 0.5,1 if "{$>%2}
" rotate3d[-1] 0,1,0,90 fi add3d[-1] 70 add3d rotate3d 0,0,1,18 done moded3d
```



### Create a vase from a 3D isosurface:

```
$ gmic moded3d 4 isosurface3d "'x^2+2*abs(y/2)*sin(2*y)^2+z^2-3',
0" sphere3d 1.5 sub3d[-1] 0,5 plane3d 15,15 rotate3d[-1] 1,0,0,90 center3d[-
```



### Launch a set of interactive demos:

```
$ gmic demos
```

---

## abs

**Built-in command**

**No arguments**

### Description:

Compute the pointwise absolute values of selected images.

### Examples of use:

- **Example #1**

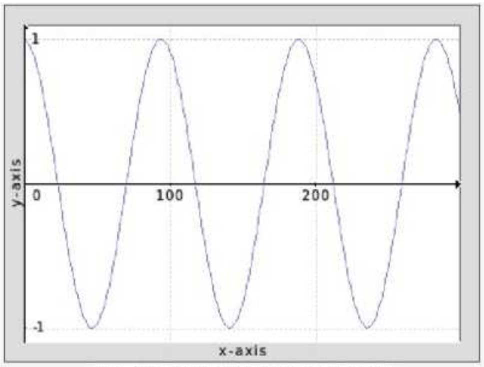
```
image.jpg +sub {ia} abs[-1]
```

[0]: 'image.jpg' (640x427x1x3)

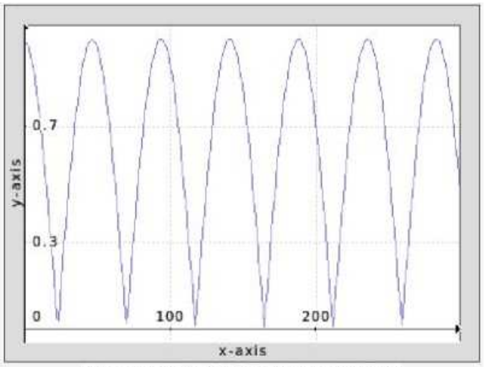
[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
300,1,1,1,'cos(20*x/w)' +abs display_graph 400,300
```



[0]: '[cos(20*x/w)]' (400x300x1x3)



[1]: '[cos(20*x/w)]_c1' (400x300x1x3)

# acos

**Built-in command**

### No arguments

## Description:

Compute the pointwise arccosine of selected images.

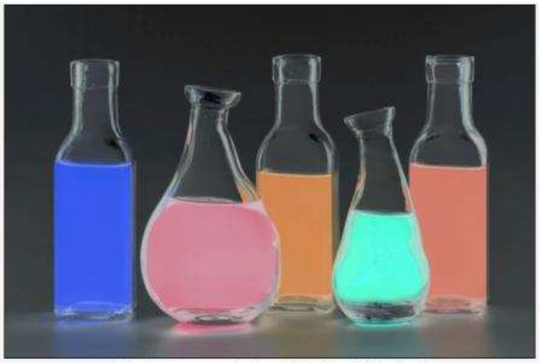
This command has a **tutorial page**.

## Examples of use:

- **Example #1**

```
image.jpg +normalize -1,1 acos[-1]
```

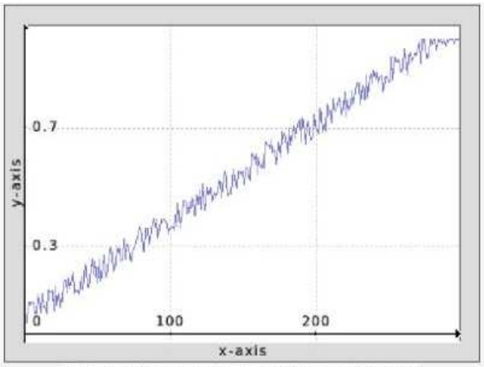
[0]: 'image.jpg' (640x427x1x3)



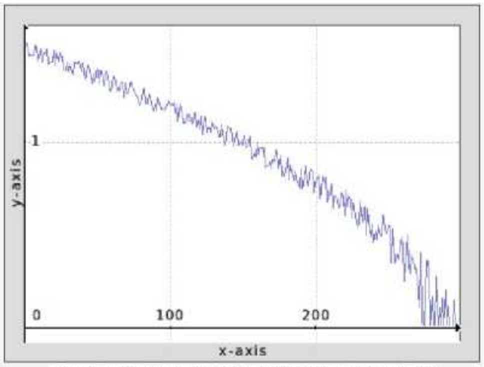
[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
300,1,1,1,'cut(x/w+0.1*u,0,1)' +acos display_graph 400,300
```



[0]: '[cut(x/w+0.1*u,0,1)]' (400x300x1x3)



[1]: '[cut(x/w+0_c1.1*u,0,1)]' (400x300x1x3)

# acosh

**Built-in command**

### No arguments

## Description:

Compute the pointwise hyperbolic arccosine of selected images.

# add

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

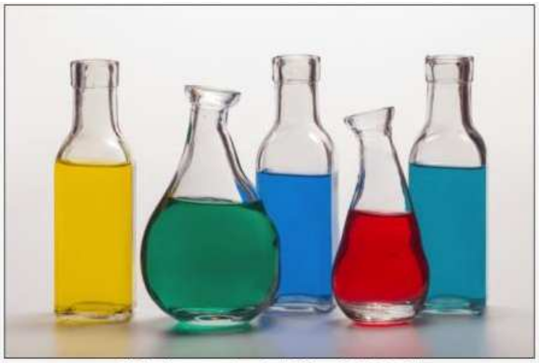
Add specified value, image or mathematical expression to selected images, or compute the pointwise sum of selected images.

(*equivalent to shortcut command* `+`).

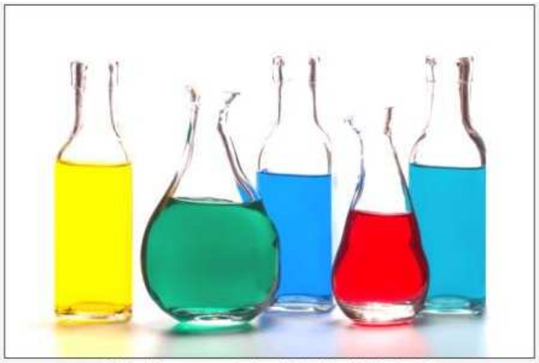
## Examples of use:

### • Example #1

```
image.jpg +add 30% cut 0,255
```



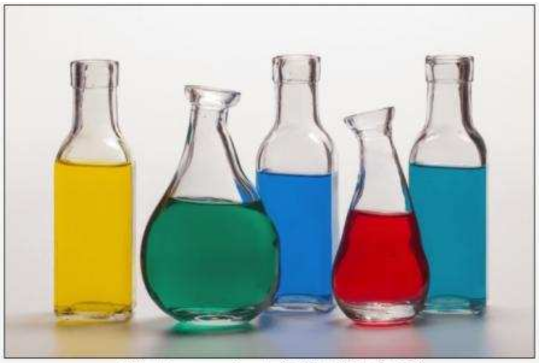
[0]: 'image.jpg' (640x427x1x3)



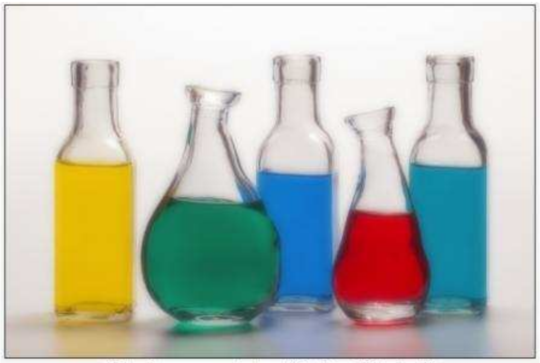
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +blur 5 normalize 0,255 add[1] [0]
```



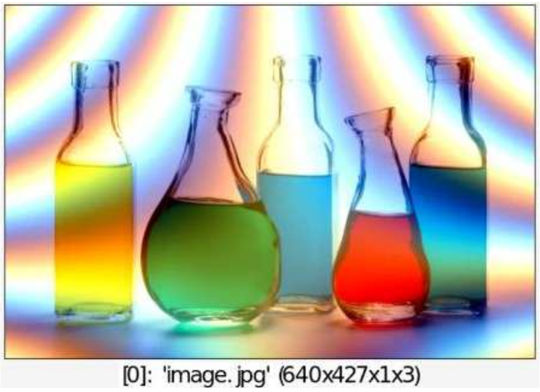
[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

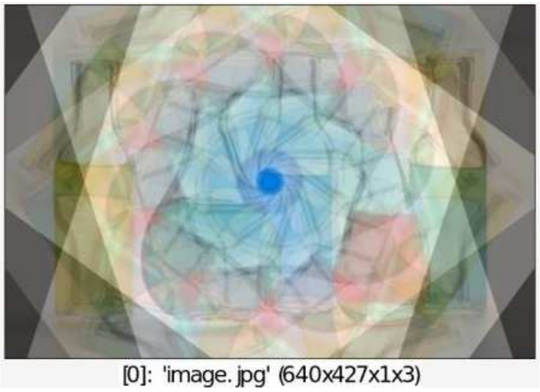
### • Example #3

```
image.jpg add '80*cos(80*(x/w-0.5)*(y/w-0.5)+c)' cut 0,255
```


[0]: 'image.jpg' (640x427x1x3)

- **Example #4**

```
image.jpg repeat 9 { +rotate[0] {$>*36},1,0,50%,50% } add div 10
```


[0]: 'image.jpg' (640x427x1x3)

---

# add3d

**Built-in command**

## Arguments:

- `tx,_ty,_tz` or
- `[object3d]` or
- `(no arg)`

## Description:

Shift selected 3D objects with specified displacement vector, or merge them with specified

3D object, or merge all selected 3D objects together.

(*equivalent to shortcut command* `+3d`).

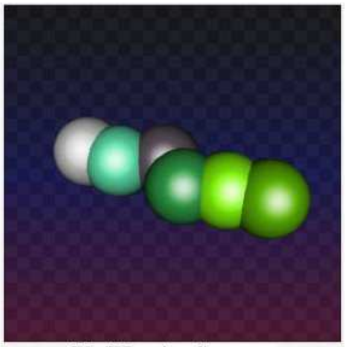
## Default values:

`ty=tz=0`.

## Examples of use:

- **Example #1**

```
sphere3d 10 repeat 5 { +add3d[-1] 10,{u(-10,10)},0 color3d[-1] ${-
rgb} } add3d
```



[0]: '[3D sphere]'
(3852 vert., 7680 prim.)

- **Example #2**

```
repeat 20 { torus3d 15,2 color3d[-1] ${-rgb} mul3d[-1] 0.5,1 if $>%2
rotate3d[-1] 0,1,0,90 fi add3d[-1] 70 add3d rotate3d[-1] 0,0,1,18 }
double3d 0
```

[0]: '[3D torus]' (5760 vert., 5760 prim.)

# add_copymark

### No arguments

## Description:

Add copymark suffix in names of selected images.

# adjust_colors

## Arguments:

- `-100<=_brightness<=100,-100<=_contrast<=100,-100<=_gamma<=100,-100<=_hue_shift<=100,-100<=_saturation<=100,_value_min,_value_max`

## Description:

Perform a global adjustment of colors on selected images.

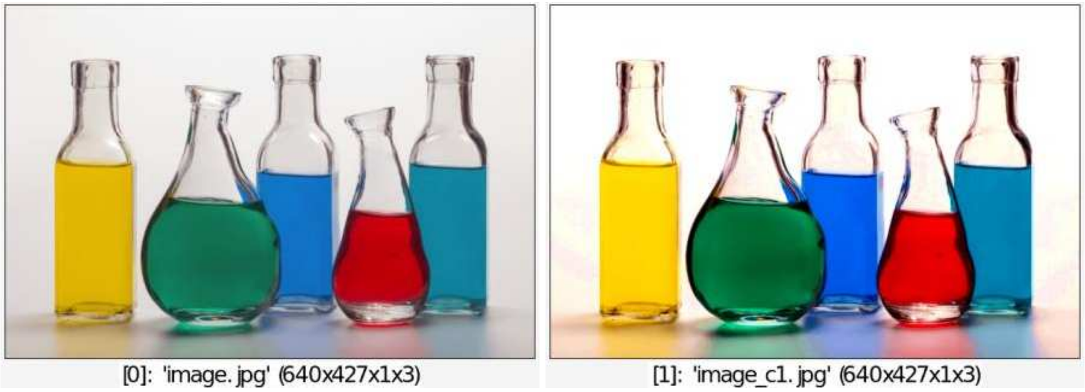
Range of correct image values are considered to be in [value_min,value_max] (e.g. [0,255]).
If `value_min==value_max==0`, value range is estimated from min/max values of selected images.
Processed images have pixel values constrained in [value_min,value_max].

## Default values:

`brightness=0`, `contrast=0`, `gamma=0`, `hue_shift=0`, `saturation=0`, `value_min=value_max=0`.

## Example of use:

```
image.jpg +adjust_colors 0,30,0,0,30
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

---

# alert

## Arguments:

- `_title,_message,_label_button1,_label_button2,...`

## Description:

Display an alert box and wait for user's choice.

If a single image is in the selection, it is used as an icon for the alert box.

## Default values:

'title=[G'MIC Alert]' and 'message=This is an alert box.'.

---

# and

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

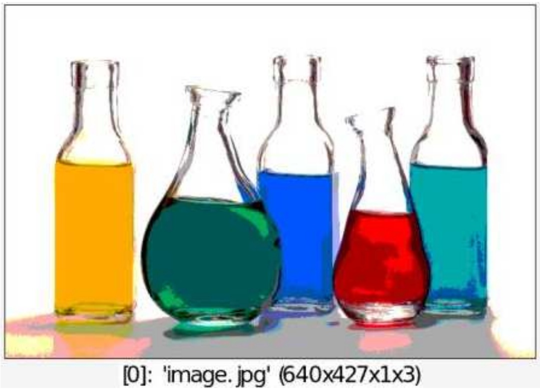
Compute the bitwise AND of selected images with specified value, image or mathematical expression, or compute the pointwise sequential bitwise AND of selected images.

(*equivalent to shortcut command* `&`).

## Examples of use:

### • Example #1

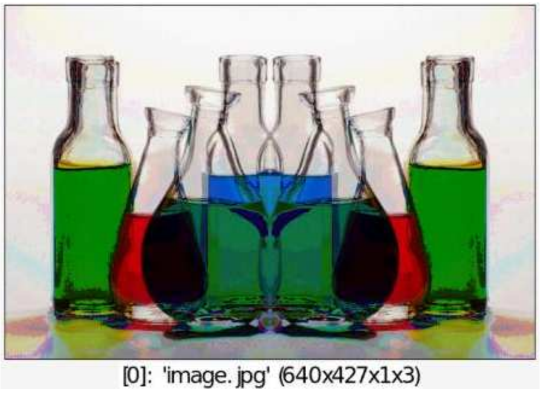
```
image.jpg and {128+64}
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +mirror x and
```



[0]: 'image.jpg' (640x427x1x3)

# animate

## Arguments:

- `filter_name,"param1_start,...,paramN_start","param1_end,...,paramN_end",nb_fr 0 | 1 },_output_filename` or
- `delay>0,_back and forth={ 0 | 1 }`

## Description:

Animate filter from starting parameters to ending parameters or animate selected images

in a display window.

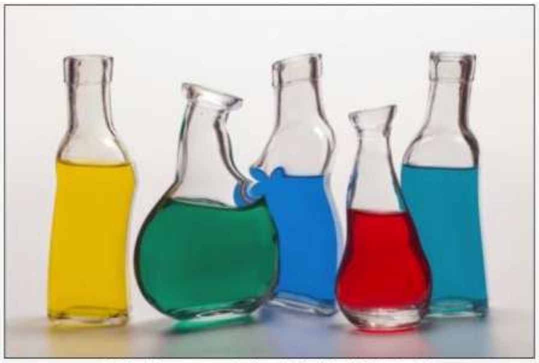
## Default values:

`delay=30`.

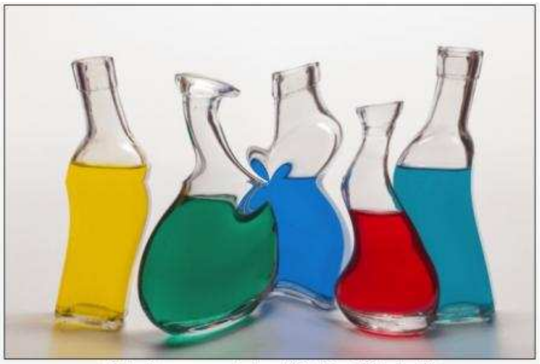
## Example of use:

```
image.jpg animate flower,"0,3","20,8",9
```

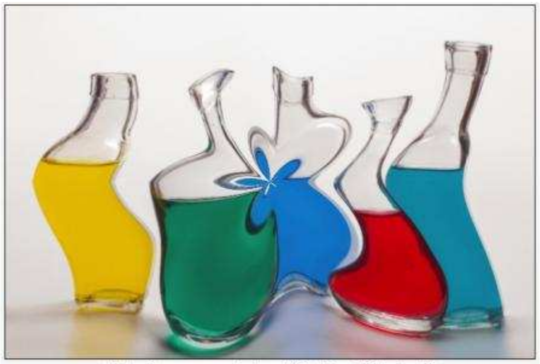
[0]: 'image_c1.jpg' (640x427x1x3)



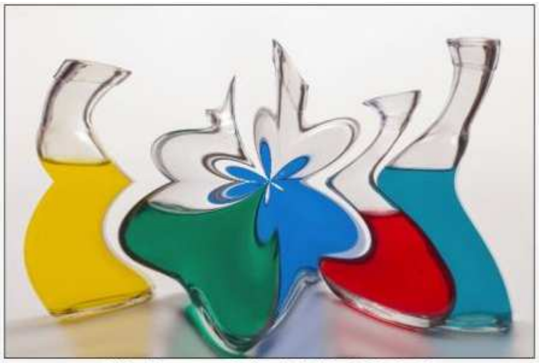
[1]: 'image_c1.jpg' (640x427x1x3)



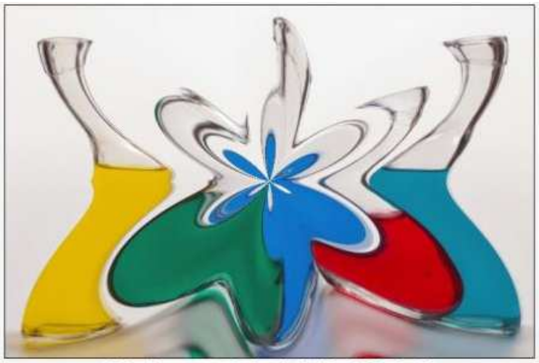
[2]: 'image_c1.jpg' (640x427x1x3)



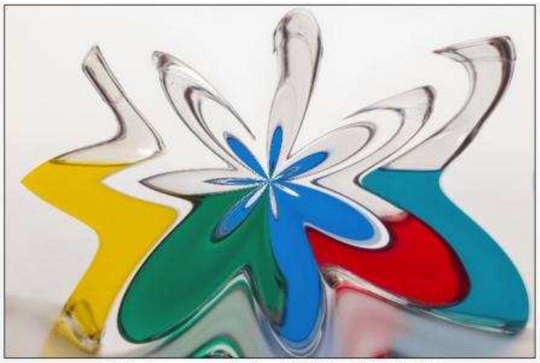
[3]: 'image_c1.jpg' (640x427x1x3)



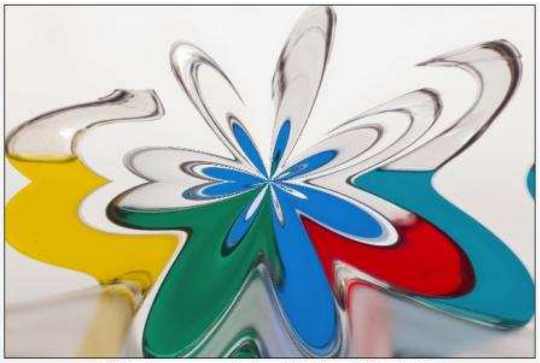
[4]: 'image_c1.jpg' (640x427x1x3)



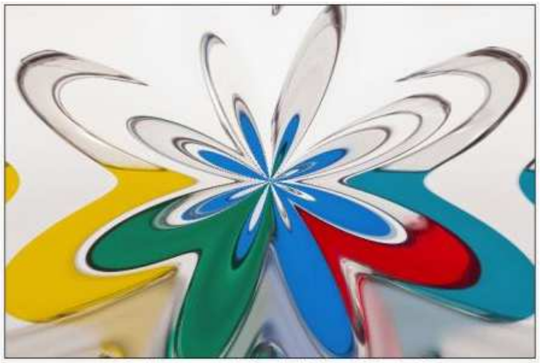
[5]: 'image_c1.jpg' (640x427x1x3)



[6]: 'image_c1.jpg' (640x427x1x3)



[7]: 'image_c1.jpg' (640x427x1x3)



[8]: 'image_c1.jpg' (640x427x1x3)

# animate3d

## Arguments:

- `nb_frames>0,_step_angle_x,_step_angle_y,_step_angle_z,_zoom_factor,0<=_fake_shadow_level<=100,_[background]`

## Description:

Generate 3D animation frames of rotating 3D objects.

Frames are stacked along the z-axis (volumetric image).
Frame size is the same as the size of the `[background]` image (or 800x800 if no background specified).

## Default values:

`step_angle_x=0`, `step_angle_y=5`, `step_angle_z=0`, `zoom_factor=1`, `fake_shadow_level=50` and `background=(undefined)`.

# append

Built-in command

## Arguments:

- `[image],axis,_centering` or

- `axis,_centering`

## Description:

Append specified image to selected images, or all selected images together, along specified axis.

(*equivalent to shortcut command* `a`).

`axis` can be ***{ x | y | z | c }***.
Usual `centering` values are ***{ 0:left-justified | 0.5:centered | 1:right-justified }***.

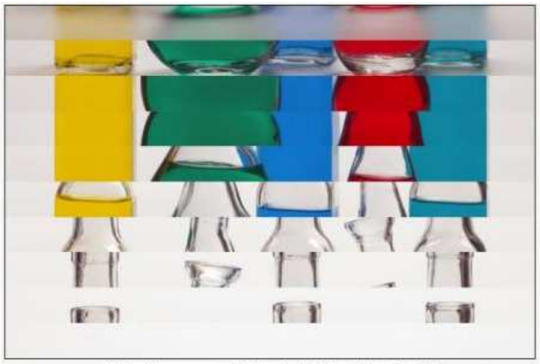
## Default values:

`centering=0`.

## Examples of use:

### • Example #1

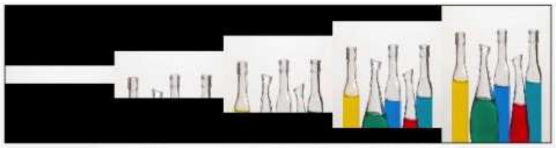
```
image.jpg split y,10 reverse append y
```



[0]: 'image_c9.jpg' (640x427x1x3)

### • Example #2

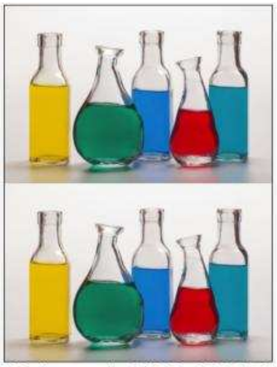
```
image.jpg repeat 5 { +rows[0] 0,{10+18*$>}% } remove[0] append x,0.5
```



[0]: 'image_c1.jpg' (3200x350x1x3)

### • Example #3

```
image.jpg append[0] [0],y
```



[0]: 'image.jpg' (640x854x1x3)

---

# append_tiles

## Arguments:

- `_M>=0,_N>=0,0<=_centering_x<=1,0<=_centering_y<=1`

## Description:

Append MxN selected tiles as new images.

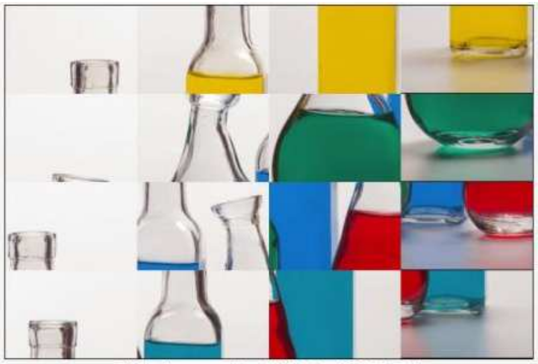
If `N` is set to 0, number of rows is estimated automatically.
If `M` is set to 0, number of columns is estimated automatically.
If `M` and `N` are both set to `0`, auto-mode is used.
If `M` or `N` is set to 0, only a single image is produced.
`centering_x` and `centering_y` tells about the centering of tiles when they have different sizes.

## Default values:

`M=0`, `N=0`, `centering_x=centering_y=0.5`.

## Example of use:

```
image.jpg split xy,4 append_tiles ,
```



[0]: 'image.jpg' (640x428x1x3)

---

# apply_camera

## Arguments:

- `_"command",_camera_index>=0,_skip_frames>=0,_output_filename`

## Description:

Apply specified command on live camera stream, and display it on display window [0].

This command requires features from the OpenCV library (not enabled in G'MIC by default).

## Default values:

`command=""`, `camera_index=0` (default camera), `skip_frames=0` and `output_filename=""`.

---

# apply_camera3d

## Arguments:

- `pos_x,pos_y,pos_z,target_x,target_y,target_z,up_x,up_y,up_z`

## Description:

Apply 3D camera matrix to selected 3D objects.

## Default values:

`target_x=0`, `target_y=0`, `target_z=0`, `up_x=0`, `up_y=-1` and `up_z=0`.

---

# apply_channels

## Arguments:

- `"command",color_channels,_value_action={ 0:none | 1:cut | 2:normalize }`

## Description:

Apply specified command on the chosen color channel(s) of each selected images.

(*equivalent to shortcut command* `ac`).

Argument `color_channels` refers to a colorspace, and can be basically one of ***{ all | rgba | [s]rgb | ryb | lrgb | ycbcr | lab | lch | hsv | hsi | hsl | cmy | cmyk | yiq }***.

You can also make the processing focus on a few particular channels of this colorspace, by setting `color_channels` as `colorspace_channel` (e.g. `hsv_h` for the hue).
All channel values are considered to be provided in the [0,255] range.

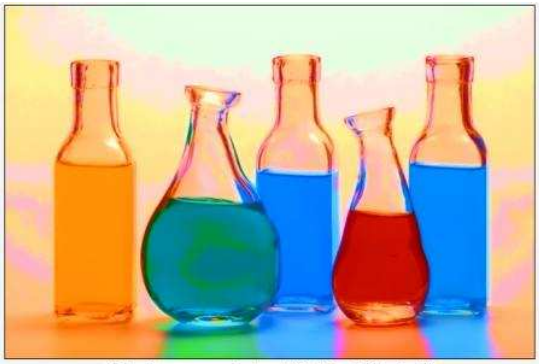
### Default values:

`value_action=0`.

### Example of use:

```
image.jpg +apply_channels "equalize blur 2",ycbcr_cbcr
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## apply_curve

### Arguments:

- `0<=smoothness<=1,x0,y0,x1,y1,x2,y2,...,xN,yN`

### Description:

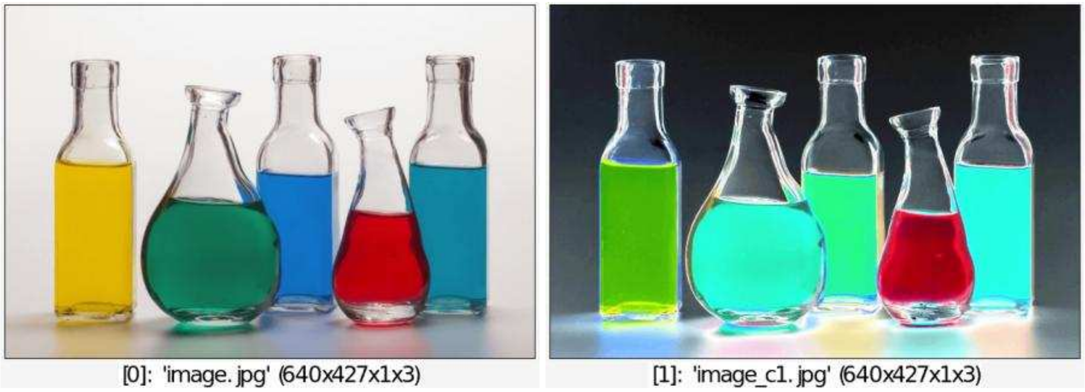
Apply curve transformation to image values.

### Default values:

`smoothness=1`, `x0=0`, `y0=100`.

### Example of use:

```
image.jpg +apply_curve 1,0,0,128,255,255,0
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

# apply_files

## Arguments:

- `"filename_pattern",_"command",_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1,_output_filename`

## Description:

Apply a G'MIC command on specified input image files, in a streamed way.

If a display window is opened, rendered frames are displayed in it during processing.
The output filename may have extension `.avi` or `.mp4` (saved as a video), or any other usual image file
extension (saved as a sequence of images).

## Default values:

`command=(undefined)`, `first_frame=0`, `last_frame=-1`, `frame_step=1` and `output_filename=(undefined)`.

# apply_gamma

## Arguments:

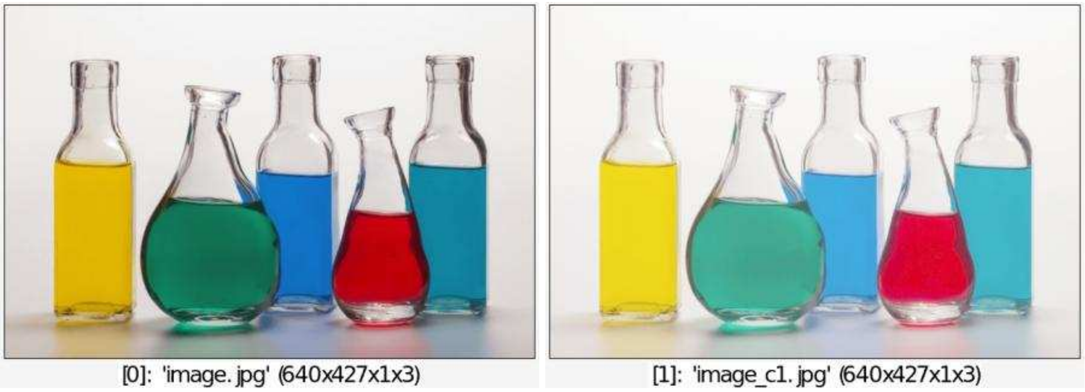
- `gamma>=0`

## Description:

Apply gamma correction to selected images.

## Example of use:

```
image.jpg +apply_gamma 2
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

# apply_matrix3d

## Arguments:

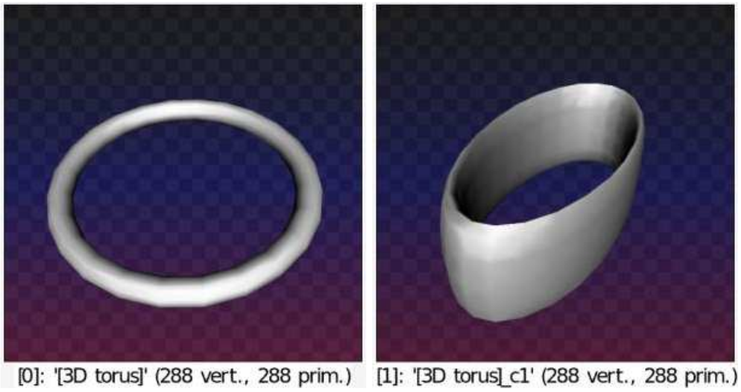
- `a11,a12,a13,...,a31,a32,a33`

## Description:

Apply specified 3D rotation matrix to selected 3D objects.

## Example of use:

```
torus3d 10,1 +apply_matrix3d {mul(rot(1,0,1,-15°),
[1,0,0,0,2,0,0,0,8],3)} double3d 0
```


[0]: '[3D torus]' (288 vert., 288 prim.)
[1]: '[3D torus]_c1' (288 vert., 288 prim.)

# apply_parallel

## Arguments:

- `"command"`

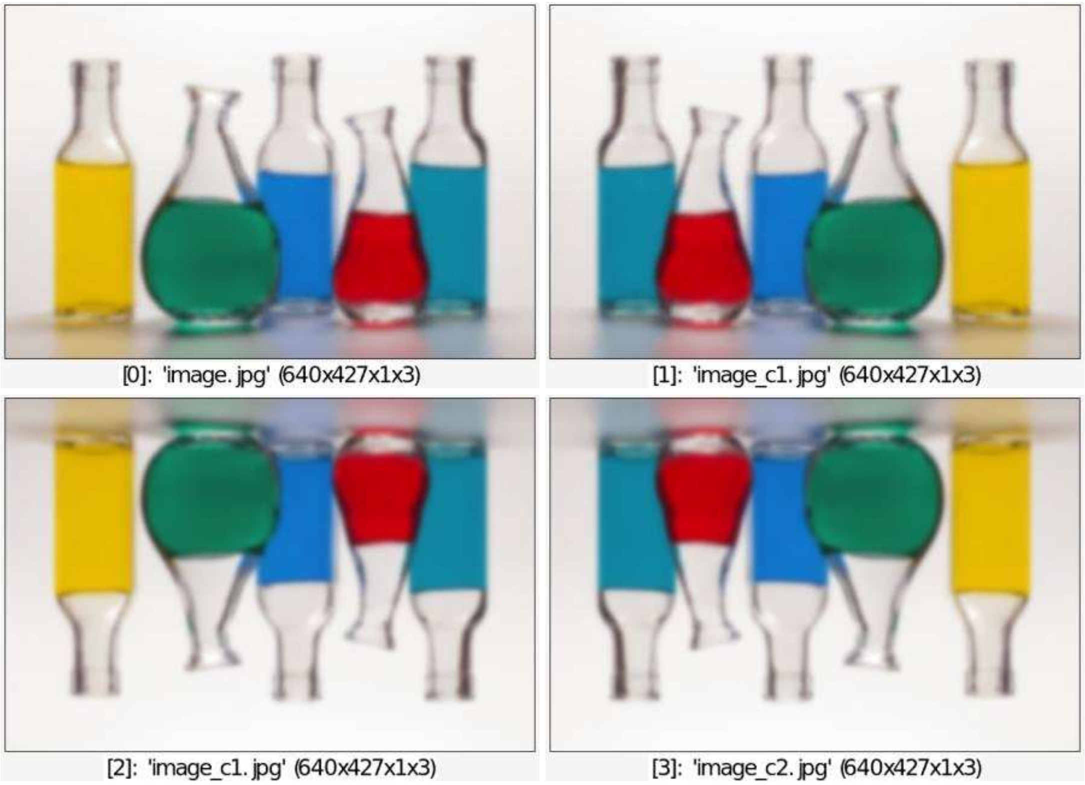
## Description:

Apply specified command on each of the selected images, by parallelizing it for all image of the list.

(*equivalent to shortcut command* `ap`).

## Example of use:

```
image.jpg +mirror x +mirror y apply_parallel "blur 3"
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c1.jpg' (640x427x1x3)

[3]: 'image_c2.jpg' (640x427x1x3)

# apply_parallel_channels

## Arguments:

- `"command"`

## Description:

Apply specified command on each of the selected images, by parallelizing it for all channel of the images independently.

(*equivalent to shortcut command* `apc`).

## Example of use:

```
image.jpg apply_parallel_channels "blur 3"
```

[0]: 'image.jpg' (640x427x1x3)

# apply_parallel_overlap

## Arguments:

- `"command",overlap[%],nb_threads={ 0:auto | 1 | 2 | 4 | 8 | 16 }`

## Description:

Apply specified command on each of the selected images, by parallelizing it on `nb_threads` overlapped sub-images.

(*equivalent to shortcut command* `apo`).

`nb_threads` must be a power of 2.

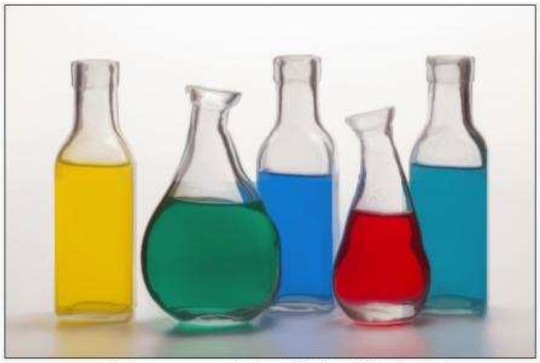
## Default values:

`overlap=0`, `nb_threads=0`.

## Example of use:

```
image.jpg +apply_parallel_overlap "smooth 500,0,1",1
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# apply_scales

## Arguments:

- `"command",number_of_scales>0,_min_scale[%]>=0,_max_scale[%]>=0,_scale_gamma>0`

## Description:

Apply specified command on different scales of selected images.

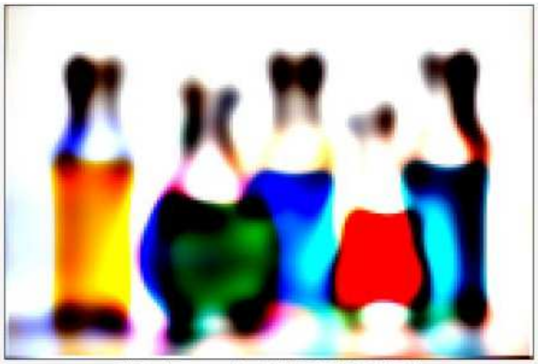
`interpolation` can be ***`{ 0:none | 1:nearest | 2:average | 3:linear | 4:grid | 5:bicubic | 6:lanczos }`***.

## Default values:

`min_scale=25%`, `max_scale=100%` and `interpolation=3`.

## Example of use:

```
image.jpg apply_scales "blur 5 sharpen 1000",4
```



[0]: 'image.jpg' (160x107x1x3)

[1]: 'image.jpg' (320x214x1x3)

[2]: 'image.jpg' (480x320x1x3)

[3]: 'image.jpg' (640x427x1x3)

# apply_tiles

## Arguments:

- `"command",_tile_width[%]>0,_tile_height[%]>0,_tile_depth[%]>0,_overlap_width[`
`0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Apply specified command on each tile (neighborhood) of the selected images, eventually with overlapping tiles.

(*equivalent to shortcut command* `at`).

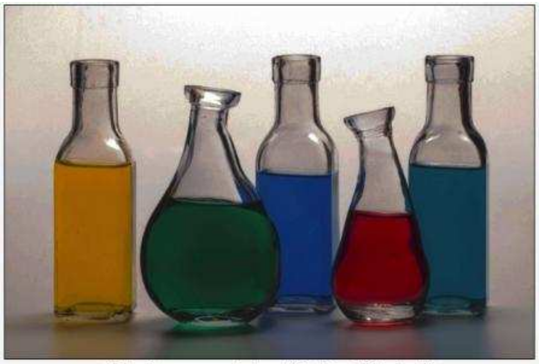
## Default values:

`tile_width=tile_height=tile_depth=10%`, `overlap_width=overlap_height=overlap_depth=0` and `boundary_conditions=1`.

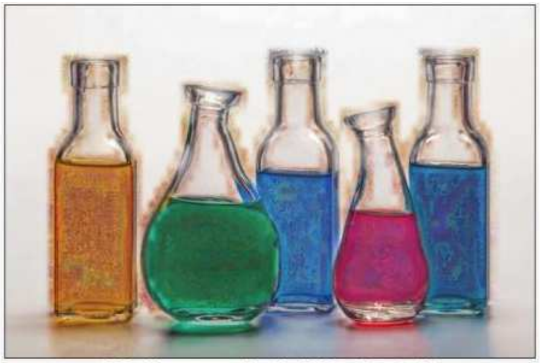
## Example of use:

```
image.jpg +equalize[0] 256 +apply_tiles[0] "equalize 256",
16,16,1,50%,50%
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: '[unnamed]_c1' (640x427x1x3)

# apply_timeout

## Arguments:

- `"command",_timeout={ 0:no timeout | >0:with specified timeout (in seconds) }`

## Description:

Apply a command with a timeout.

Set variable `$_is_timeout` to `1` if timeout occurred, `0` otherwise.

## Default values:

`timeout=20`.

---

# apply_video

## Arguments:

- `video_filename,_"command",_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1,_output_filename`

## Description:

Apply a G'MIC command on all frames of the specified input video file, in a streamed way.

If a display window is opened, rendered frames are displayed in it during processing.
The output filename may have extension `.avi` or `.mp4` (saved as a video), or any other usual image
file extension (saved as a sequence of images).
This command requires features from the OpenCV library (not enabled in G'MIC by default).

## Default values:

`first_frame=0`, `last_frame=-1`, `frame_step=1` and `output_filename=(undefined)`.

---

# area

## Arguments:

- `tolerance>=0,is_high_connectivity={ 0 | 1 }`

## Description:

Compute area of connected components in selected images.

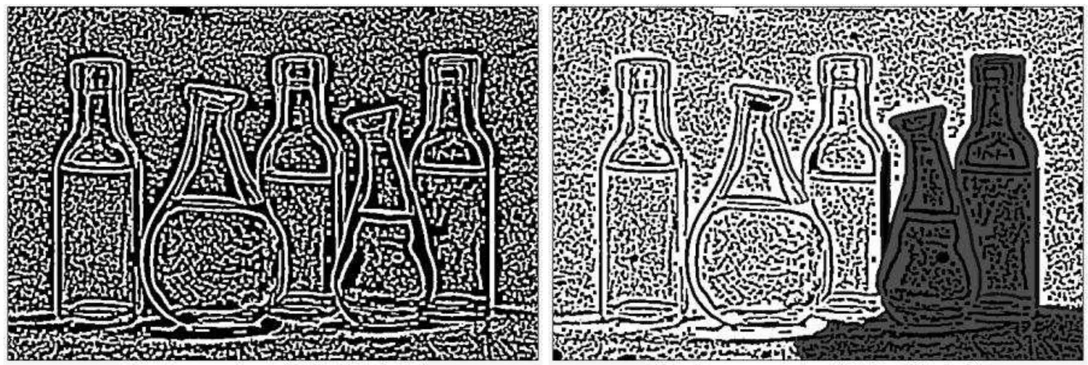
## Default values:

`is_high_connectivity=0`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg luminance stencil[-1] 1 +area 0
```



[0]: 'image.jpg' (640x427x1x1) [1]: 'image_c1.jpg' (640x427x1x1)

## area_fg

### Arguments:

- `tolerance>=0,is_high_connectivity={ 0 | 1 }`

### Description:

Compute area of connected components for non-zero values in selected images.

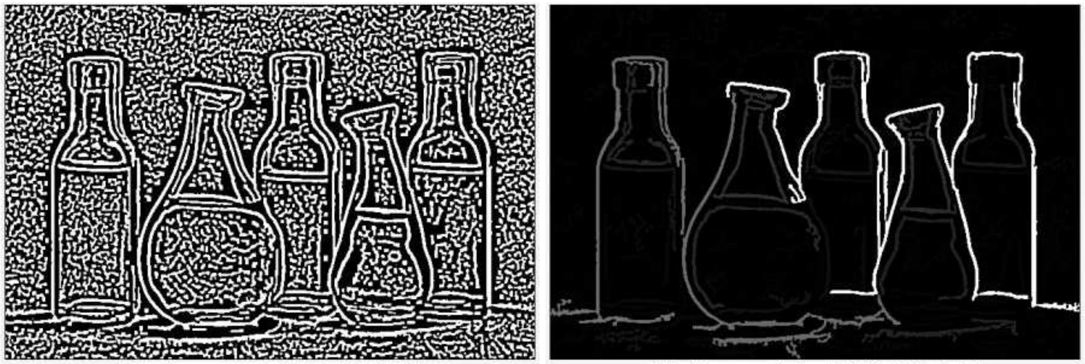
Similar to `area` except that 0-valued pixels are not considered.

### Default values:

`is_high_connectivity=0`.

### Example of use:

```
image.jpg luminance stencil[-1] 1 +area_fg 0
```



[0]: 'image.jpg' (640x427x1x1) [1]: 'image_c1.jpg' (640x427x1x1)

## arg

### Arguments:

- `n>=1,_arg1,...,_argN`

### Description:

Return the n-th argument of the specified argument list.

---

## arg0

### Arguments:

- `n>=0,_arg0,...,_argN`

### Description:

Return the n-th argument of the specified argument list (where `n` starts from `0`).

---

## arg2img

### Arguments:

- `argument_1,...,argument_N`

### Description:

Split specified list of arguments and return each as a new image (as a null-terminated string).

---

## arg2var

### Arguments:

- `variable_name,argument_1,...,argument_N`

### Description:

For each i in [1...N], set `variable_name$i=argument_i`.

The variable name should be global to make this command useful (i.e. starts by an underscore).

---

## argmax

**No arguments**

### Description:

Compute the argmax of selected images. Returns a single image

with each pixel value being the index of the input image with maximal value.

## Example of use:

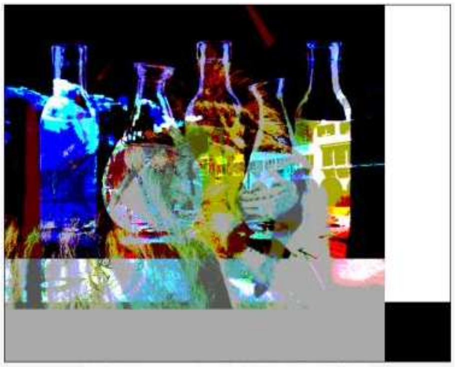
```
image.jpg sample lena,lion,square +argmax
```

[0]: 'image. jpg' (640x427x1x3)

[1]: 'lena' (512x512x1x3)

[2]: 'lion' (640x600x1x3)

[3]: 'square' (750x500x1x3)



[4]: '[argmax]_c1' (750x600x1x3)

# argmaxabs

### No arguments

## Description:

Compute the argmaxabs of selected images. Returns a single image with each pixel value being the index of the input image with maxabs value.

## argmin

**No arguments**

### Description:

Compute the argmin of selected images. Returns a single image

with each pixel value being the index of the input image with minimal value.

### Example of use:

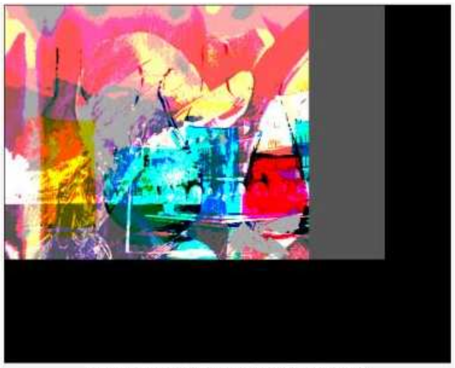
```
image.jpg sample lena,lion,square +argmin
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'lena' (512x512x1x3)

[2]: 'lion' (640x600x1x3)

[3]: 'square' (750x500x1x3)


[4]: '[argmin]_c1' (750x600x1x3)

## argminabs

### No arguments

### Description:

Compute the argminabs of selected images. Returns a single image

with each pixel value being the index of the input image with minabs value.

## array

### Arguments:

- `M>0,_N>0,_expand_type={ 0:min | 1:max | 2:all }`

### Description:

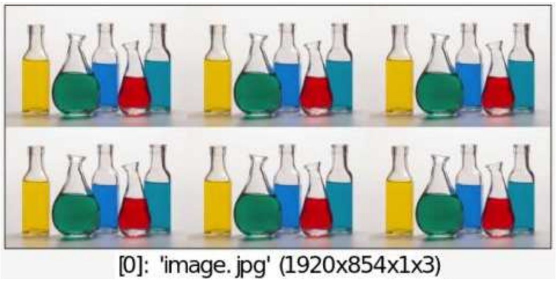
Create MxN array from selected images.

### Default values:

`N=M` and `expand_type=0`.

### Example of use:

```
image.jpg array 3,2,2
```



[0]: 'image.jpg' (1920x854x1x3)

## array3d

### Arguments:

- `size_x>=1,_size_y>=1,_size_z>=1,_offset_x[%],_offset_y[%],_offset_y[%]`

### Description:

Duplicate a 3D object along the X,Y and Z axes.

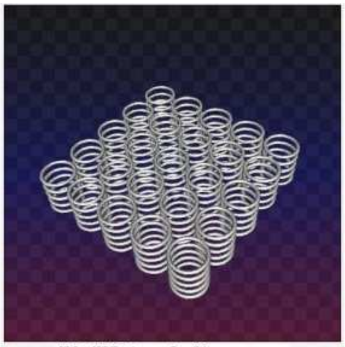
## Default values:

`size_y=1`, `size_z=1` and `offset_x=offset_y=offset_z=100%`.

## Example of use:

```
torus3d 10,1 +array3d 5,5,5,110%,110%,300%
```

[0]: '[3D torus]' (288 vert., 288 prim.)



[1]: '[3D torus]_c1'
(36000 vert., 36000 prim.)

# array_fade

## Arguments:

- `M>0,_N>0,0<=_fade_start<=100,0<=_fade_end<=100,_expand_type={0:min | 1:max | 2:all}`

## Description:

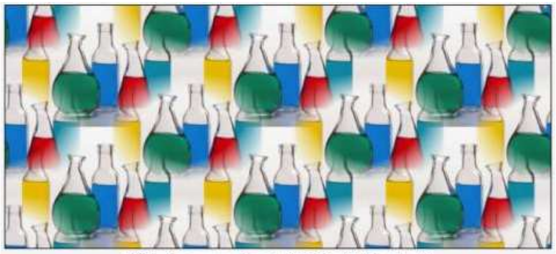
Create MxN array from selected images.

## Default values:

`N=M`, `fade_start=60`, `fade_end=90` and `expand_type=1`.

## Example of use:

```
image.jpg array_fade 3,2
```



[0]: 'image.jpg' (960x428x1x3)

## array_mirror

### Arguments:

- `N>=0,_dir={ 0:x | 1:y | 2:xy | 3:tri-xy },_expand_type={ 0 | 1 }`

### Description:

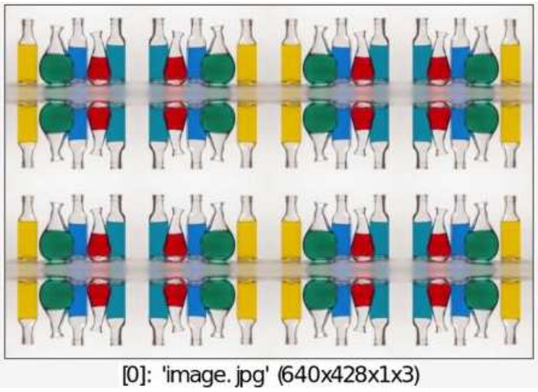
Create 2^Nx2^N array from selected images.

### Default values:

`dir=2` and `expand_type=0`.

### Example of use:

```
image.jpg array_mirror 2
```



[0]: 'image.jpg' (640x428x1x3)

## array_random

### Arguments:

- `Ms>0,_Ns>0,_Md>0,_Nd>0`

### Description:

Create MdxNd array of tiles from selected MsxNs source arrays.

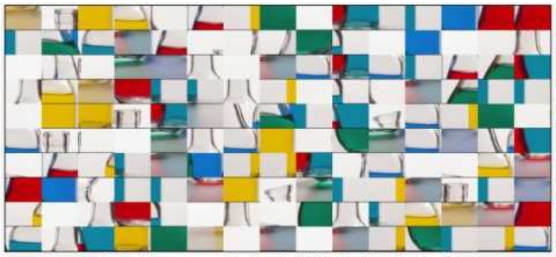
### Default values:

`Ns=Ms`, `Md=Ms` and `Nd=Ns`.

### Example of use:

```
image.jpg +array_random 8,8,15,10
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (1200x540x1x3)

# arrow

## Arguments:

- `x0[%],y0[%],x1[%],y1[%],_thickness[%]>=0,_head_length[%]>=0,_head_thickness[%`
  `...`

## Description:

Draw specified arrow on selected images.

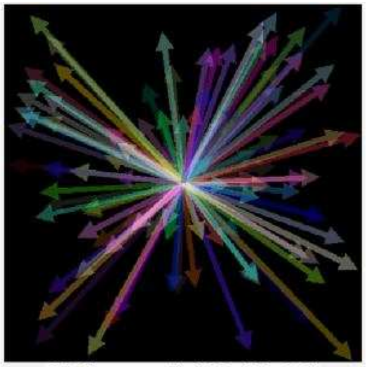
`pattern` is an hexadecimal number starting with `0x` which can be omitted
even if a color is specified. If a pattern is specified, the arrow is
drawn outlined instead of filled.

## Default values:

`thickness=1%`, `head_length=10%`, `head_thickness=3%`, `opacity=1`,
`pattern=(undefined)` and `color1=0`.

## Example of use:

```
400,400,1,3 repeat 100 arrow 50%,50%,{u(100)}%,{u(100)}%,
3,20,10,0.3,${-rgb} done
```



[0]: '[unnamed]' (400x400x1x3)

# arrow3d

## Arguments:

- `x0,y0,z0,x1,y1,z1,_radius[%]>=0,_head_length[%]>=0,_head_radius[%]>=0`

## Description:

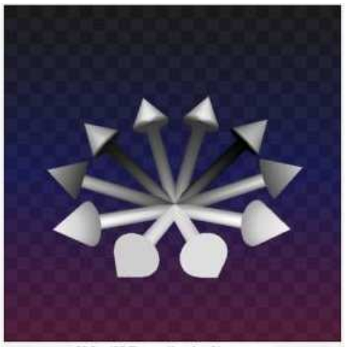
Input 3D arrow with specified starting and ending 3D points.

## Default values:

`radius=5%`, `head_length=25%` and `head_radius=15%`.

## Example of use:

```
repeat 10 { a:=$>*2*pi/10 arrow3d 0,0,0,{cos($a)},{sin($a)},-0.5 }
+3d
```



[0]: '[3D cylinder]'
(760 vert., 1200 prim.)

# asin

Built-in command

**No arguments**

## Description:

Compute the pointwise arcsine of selected images.

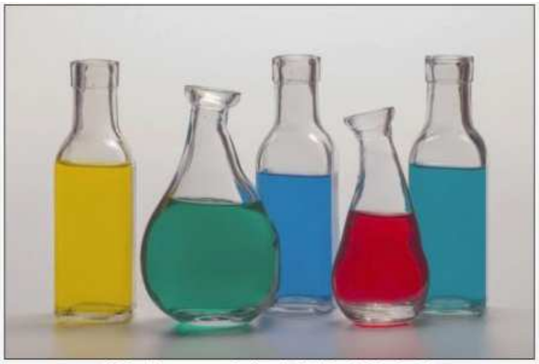
This command has a **tutorial page**.

## Examples of use:

### • Example #1

```
image.jpg +normalize -1,1 asin[-1]
```

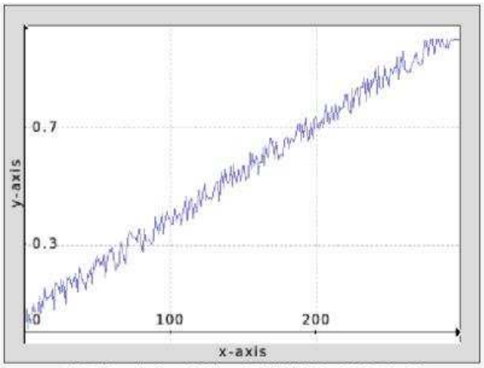
[0]: 'image.jpg' (640x427x1x3)



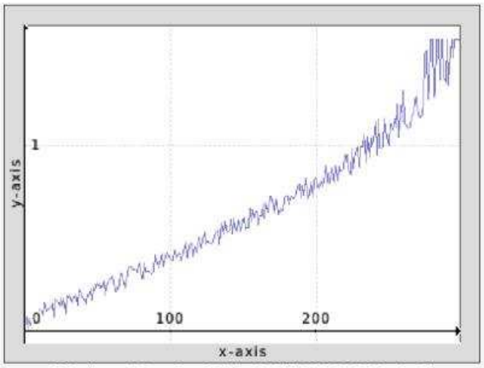
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'cut(x/w+0.1*u,0,1)' +asin display_graph 400,300
```



[0]: '[cut(x/w+0.1*u,0,1)]' (400x300x1x3)



[1]: '[cut(x/w+0_c1.1*u,0,1)]' (400x300x1x3)

# asinh

**Built-in command**

### No arguments

## Description:

Compute the pointwise hyperbolic arcsine of selected images.

# at_line

## Arguments:

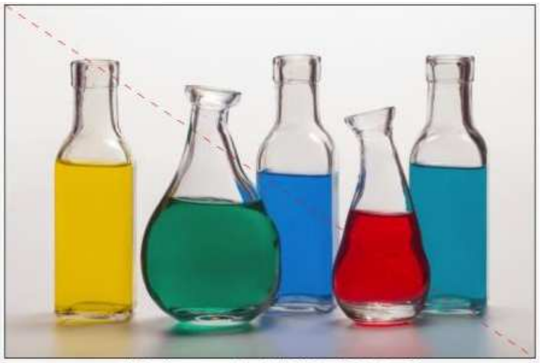
- `x0[%],y0[%],z0[%],x1[%],y1[%],z1[%]`

## Description:

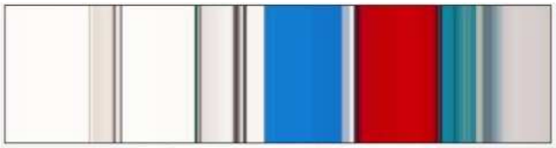
Retrieve pixels of the selected images belonging to the specified line (x0,y0,z0)-(x1,y1,z1).

## Example of use:

```
image.jpg +at_line 0,0,0,100%,100%,0 line[0] 0,0,100%,100%,
1,0xFF00FF00,255,0,0
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x1x1x3)

# at_quadrangle

## Arguments:

- `x0[%],y0[%],x1[%],y1[%],x2[%],y2[%],x3[%],y3[%],_interpolation,_boundary_cond`
  or
- `x0[%],y0[%],z0[%],x1[%],y1[%],z1[%],x2[%],y2[%],z2[%],x3[%],y3[%],z3[%],_inte`

## Description:

Retrieve pixels of the selected images belonging to the specified 2D or 3D quadrangle.

`interpolation` can be ***{ 0:nearest-neighbor | 1:linear | 2:cubic }***.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

## Example of use:

```
image.jpg params=5%,5%,95%,5%,60%,95%,40%,95% +at_quadrangle $params
polygon.. 4,$params,0.5,255
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (576x446x1x3)

# atan

Built-in command

**No arguments**

## Description:

Compute the pointwise arctangent of selected images.

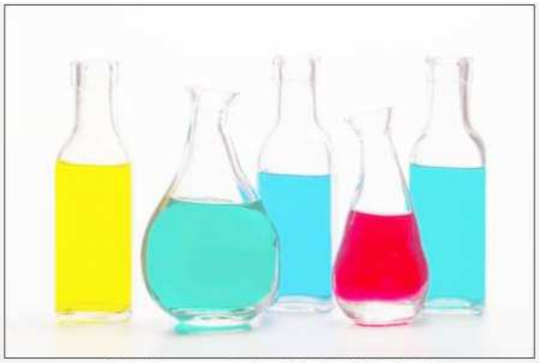
This command has a **tutorial page**.

## Examples of use:

### • Example #1

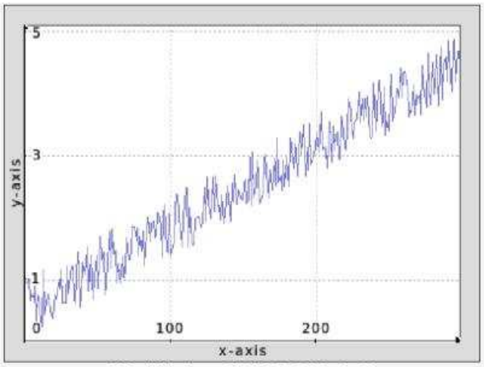
```
image.jpg +normalize 0,8 atan[-1]
```

[0]: 'image.jpg' (640x427x1x3)

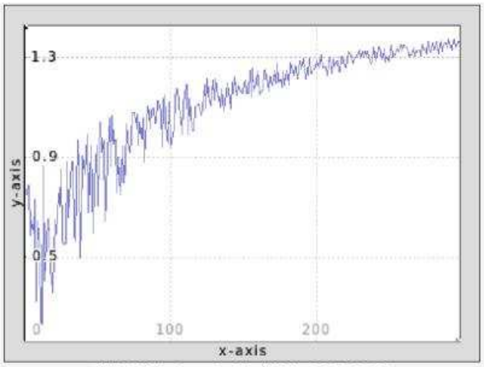

[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'4*x/w+u' +atan display_graph 400,300
```



[0]: '[4*x/w+u]' (400x300x1x3)



[1]: '[4*x/w+u]_c1' (400x300x1x3)

## atan2

Built-in command

### Arguments:

- `[x_argument]`

### Description:

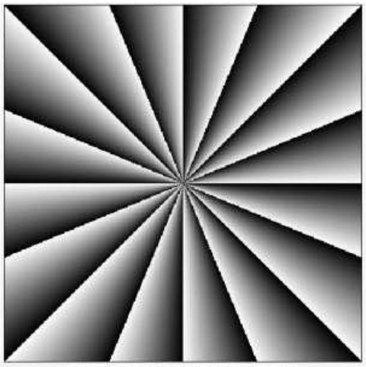
Compute the pointwise oriented arctangent of selected images.

Each selected image is regarded as the y-argument of the arctangent function, while the specified image gives the corresponding x-argument.

This command has a **tutorial page**.

### Example of use:

```
(-1,1) (-1;1) resize 400,400,1,1,3 atan2[1] [0] keep[1] mod {pi/8}
```


[0]: '(-1;1)' (400x400x1x1)

## atanh

Built-in command

**No arguments**

### Description:

Compute the pointwise hyperbolic arctangent of selected images.

# autocrop

Built-in command

## Arguments:

- `value1,value2,...` or
- `(no arg)`

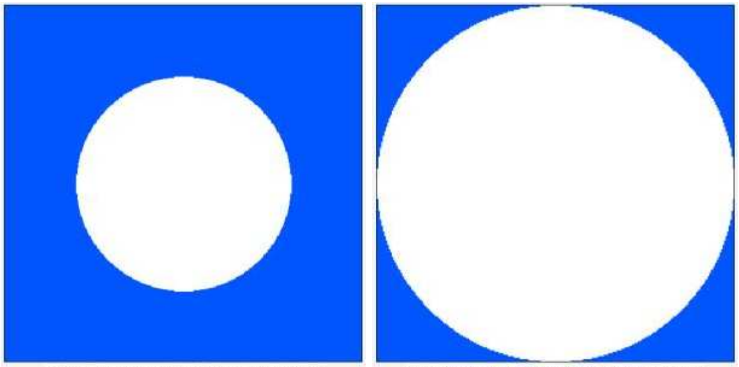
## Description:

Autocrop selected images by specified vector-valued intensity.

If no arguments are provided, cropping value is guessed.

## Example of use:

```
400,400,1,3 fill_color 64,128,255 ellipse 50%,50%,120,120,0,1,255
+autocrop
```



[0]: '[unnamed]' (400x400x1x3)    [1]: '[unnamed]_c1' (241x241x1x3)

# autocrop_components

## Arguments:

- `_threshold[%],_min_area[%]>=0,_is_high_connectivity={ 0 | 1 },_output_type={ 0:crop | 1:segmentation | 2:coordinates }`

## Description:

Autocrop and extract connected components in selected images, according to a mask given as the last channel of

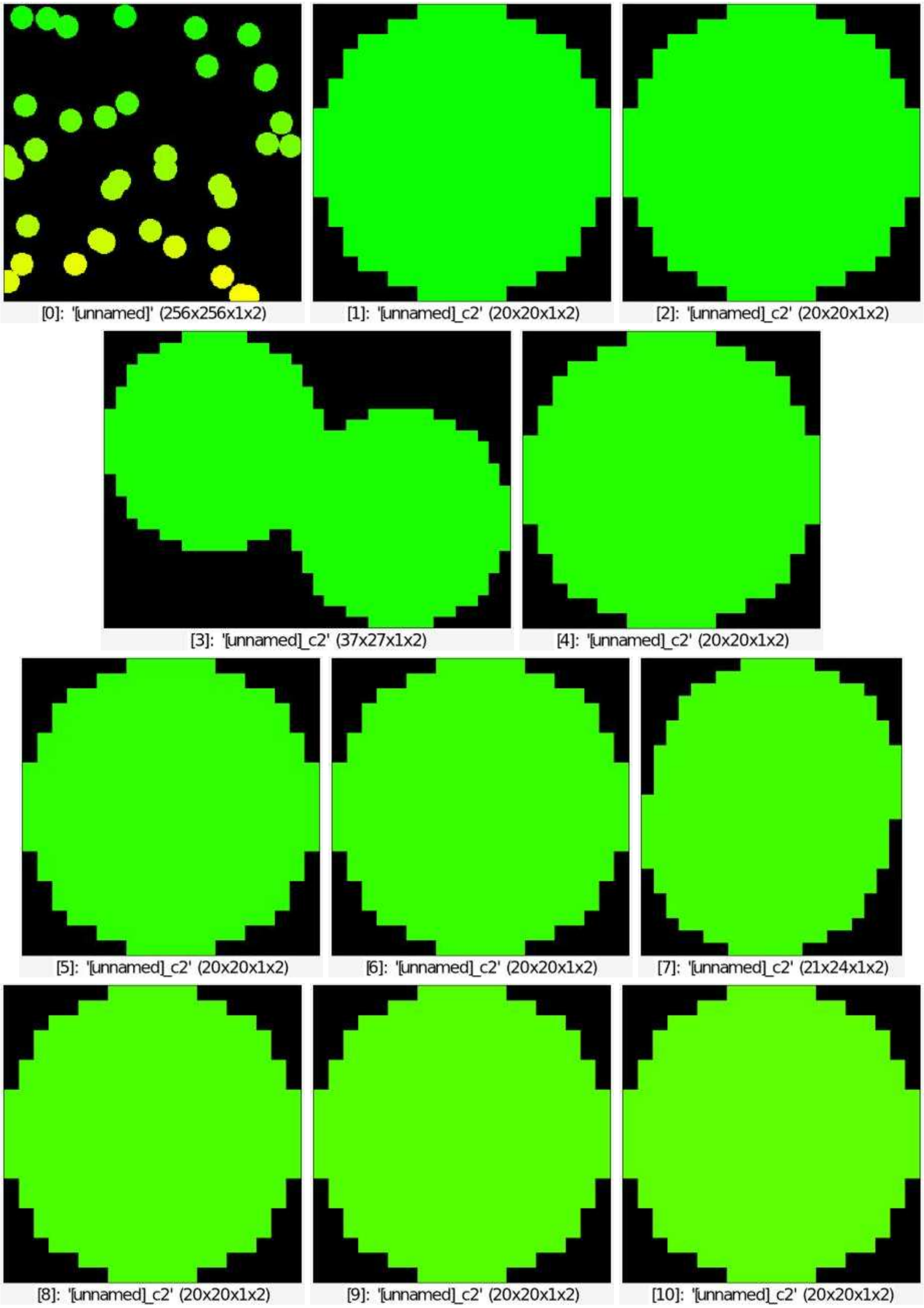
each of the selected image (e.g. alpha-channel).

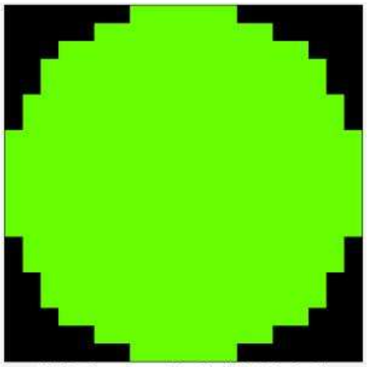
## Default values:

`threshold=0%`, `min_area=0.1%`, `is_high_connectivity=0` and `output_type=1`.

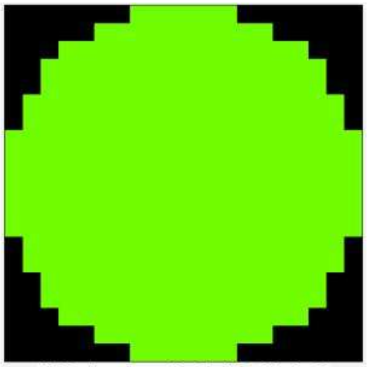
## Example of use:

```
256,256 noise 0.1,2 eq 1 dilate_circ 20 label_fg 0,1 normalize 0,255
+neq 0 *[-1] 255 append c +autocrop_components ,
```

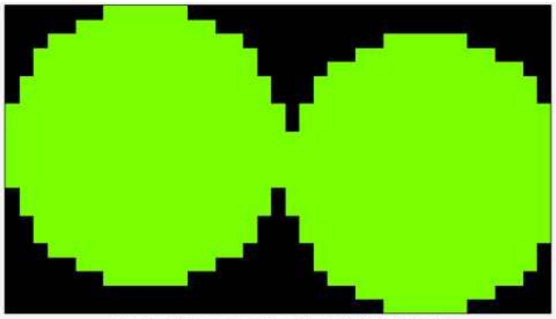

[0]: '[unnamed]' (256x256x1x2)
[1]: '[unnamed]_c2' (20x20x1x2)
[2]: '[unnamed]_c2' (20x20x1x2)
[3]: '[unnamed]_c2' (37x27x1x2)
[4]: '[unnamed]_c2' (20x20x1x2)
[5]: '[unnamed]_c2' (20x20x1x2)
[6]: '[unnamed]_c2' (20x20x1x2)
[7]: '[unnamed]_c2' (21x24x1x2)
[8]: '[unnamed]_c2' (20x20x1x2)
[9]: '[unnamed]_c2' (20x20x1x2)
[10]: '[unnamed]_c2' (20x20x1x2)



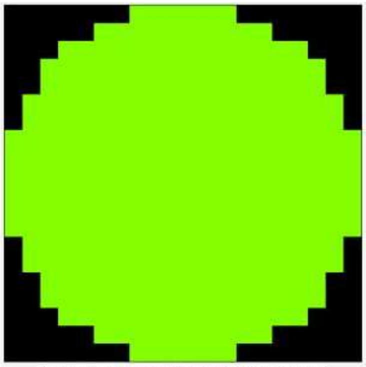
[11]: '[unnamed]_c2' (20x20x1x2)



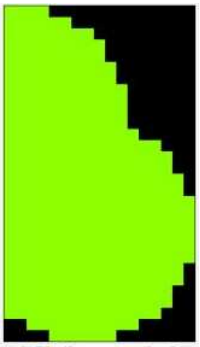
[12]: '[unnamed]_c2' (20x20x1x2)



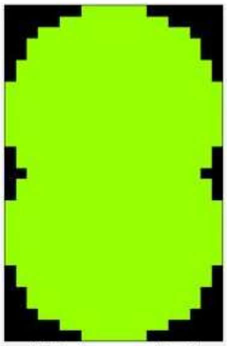
[13]: '[unnamed]_c2' (39x22x1x2)



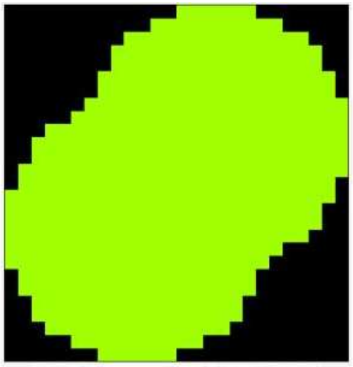
[14]: '[unnamed]_c2' (20x20x1x2)



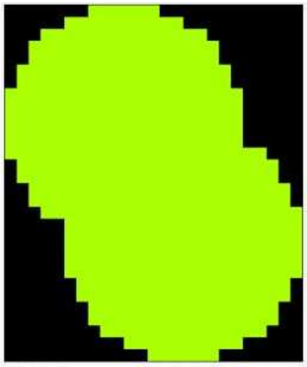
[15]: '[unnamed]_c2' (17x30x1x2)



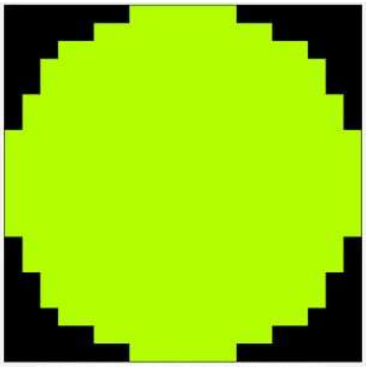
[16]: '[unnamed]_c2' (20x31x1x2)



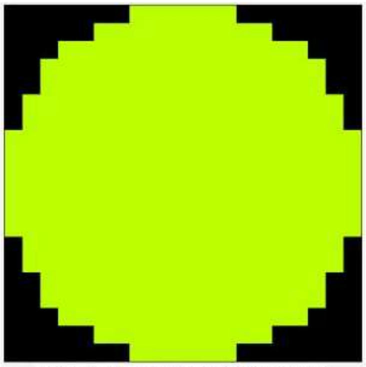
[17]: '[unnamed]_c2' (26x27x1x2)



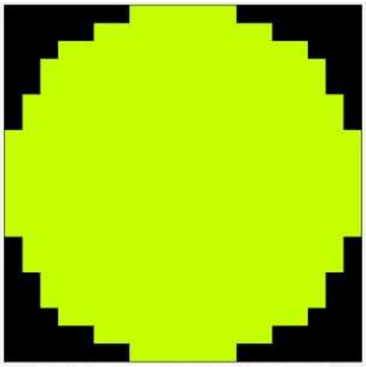
[18]: '[unnamed]_c2' (25x30x1x2)



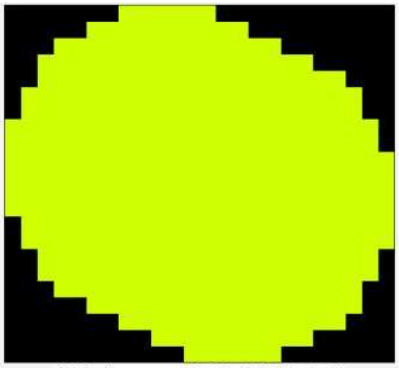
[19]: '[unnamed]_c2' (20x20x1x2)

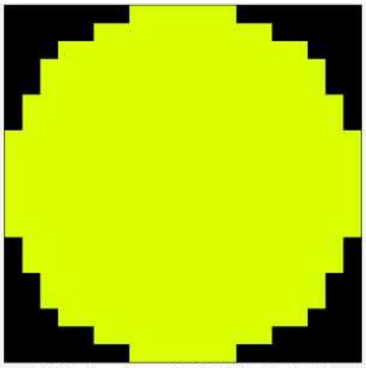


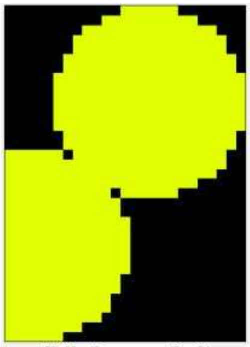
[20]: '[unnamed]_c2' (20x20x1x2)

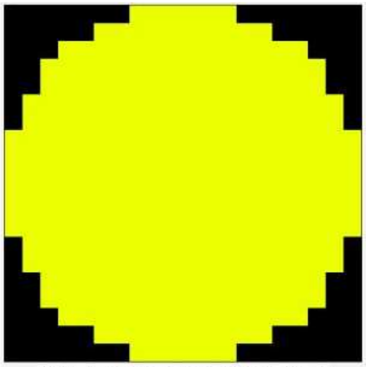


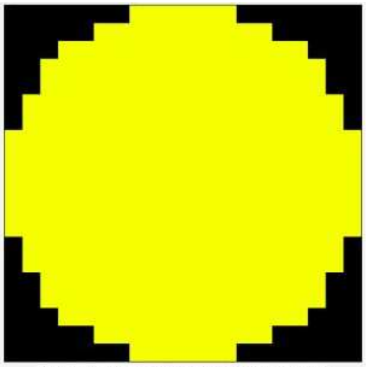
[21]: '[unnamed]_c2' (20x20x1x2)

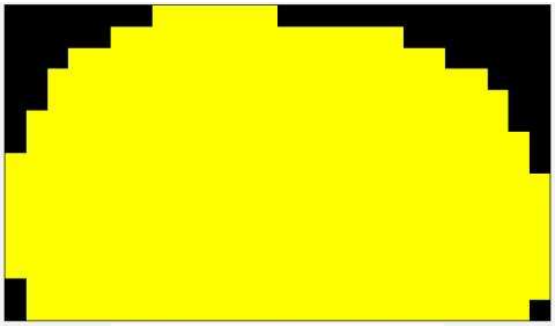

[22]: '[unnamed]_c2' (24x22x1x2)


[23]: '[unnamed]_c2' (20x20x1x2)


[24]: '[unnamed]_c2'
(25x35x1x2)


[25]: '[unnamed]_c2' (20x20x1x2)


[26]: '[unnamed]_c2' (20x20x1x2)


[27]: '[unnamed]_c2' (26x15x1x2)

# autocrop_coords

## Arguments:

- `value1,value2,... | auto`

## Description:

Return coordinates (x0,y0,z0,x1,y1,z1) of the autocrop that could be performed on the latest

of the selected images.

## Default values:

`auto`

# autocrop_seq

## Arguments:

- `value1,value2,... | auto`

## Description:

Autocrop selected images using the crop geometry of the last one by specified vector-valued intensity,

or by automatic guessing the cropping value.

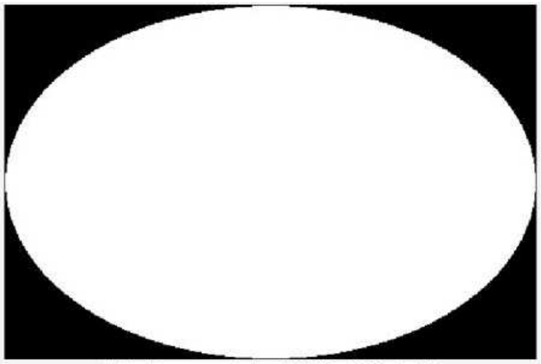
## Default values:

auto mode.

## Example of use:

```
image.jpg +fill[-1] 0 ellipse[-1] 50%,50%,30%,20%,0,1,1 autocrop_seq
0
```

[0]: 'image.jpg' (463x308x1x3)



[1]: 'image_c1.jpg' (463x308x1x3)

---

# autoindex

## Arguments:

- `nb_colors>0,0<=_dithering<=1,_method={ 0:median-cut | 1:k-means }`

## Description:

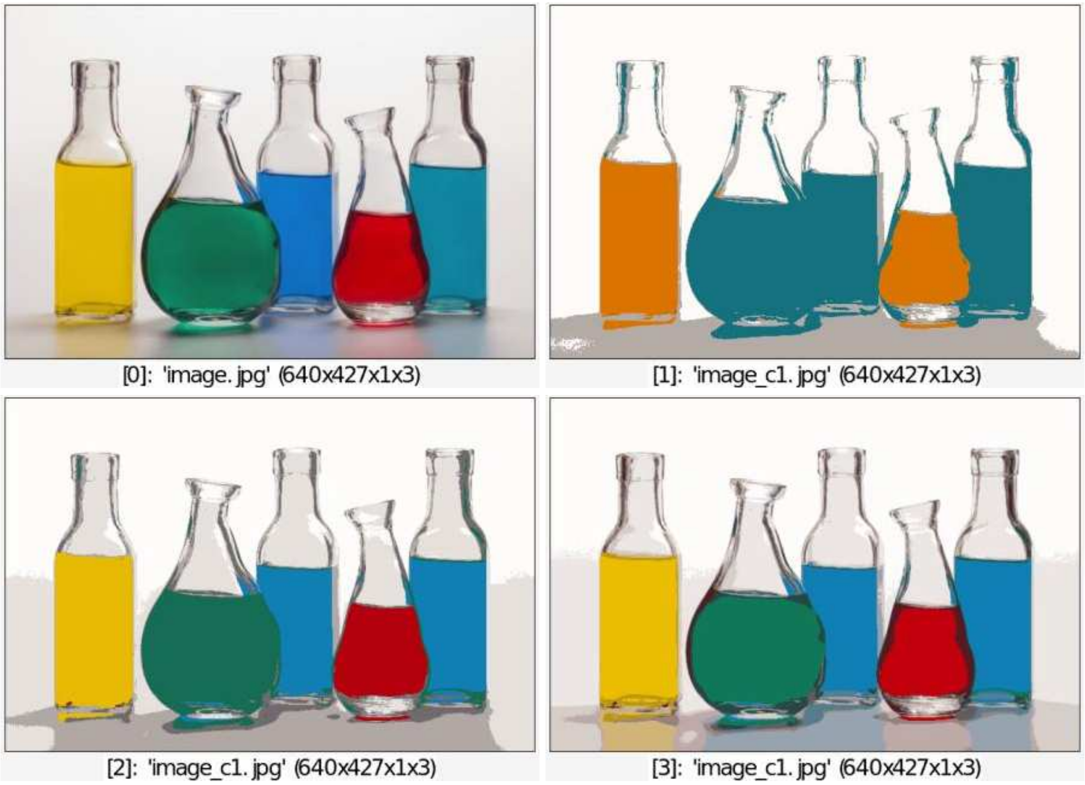
Index selected vector-valued images by adapted colormaps.

## Default values:

`dithering=0` and `method=1`.

## Example of use:

```
image.jpg +autoindex[0] 4 +autoindex[0] 8 +autoindex[0] 16
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c1.jpg' (640x427x1x3)

[3]: 'image_c1.jpg' (640x427x1x3)

# average_files

## Arguments:

- `"filename_pattern",_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1,_output_filename`

## Description:

Average specified input image files, in a streamed way.

If a display window is opened, rendered frames are displayed in it during processing.
The output filename may have extension `.avi` or `.mp4` (saved as a video), or any other usual image
file extension (saved as a sequence of images).

## Default values:

`first_frame=0`, `last_frame=-1`, `frame_step=1` and `output_filename=(undefined)`.

# average_vectors

**No arguments**

## Description:

Return the vector-valued average of the latest of the selected images.

# average_video

## Arguments:

- `video_filename,_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1,_output_filename`

## Description:

Average frames of specified input video file, in a streamed way.

If a display window is opened, rendered frames are displayed in it during processing.
The output filename may have extension `.avi` or `.mp4` (saved as a video), or any other usual image
file extension (saved as a sequence of images).
This command requires features from the OpenCV library (not enabled in G'MIC by default).

## Default values:

`first_frame=0`, `last_frame=-1`, `frame_step=1` and `output_filename=(undefined)`.

# axes

## Arguments:

- `x0,x1,y0,y1,_font_height>=0,_opacity,_pattern,_color1,...`

## Description:

Draw xy-axes on selected images.

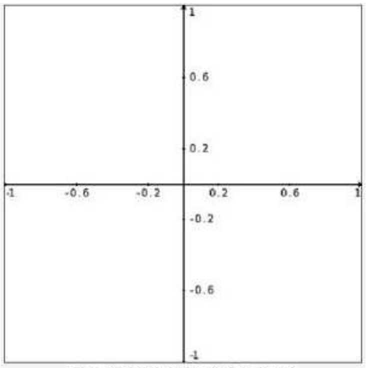
`pattern` is an hexadecimal number starting with `0x` which can be omitted
even if a color is specified.
To draw only one x-axis at row Y, set both `y0` and `y1` to Y.
To draw only one y-axis at column X, set both `x0` and `x1` to X.

## Default values:

`font_height=14`, `opacity=1`, `pattern=(undefined)` and `color1=0`.

## Example of use:

```
400,400,1,3,255 axes -1,1,1,-1
```



[0]: '[255]' (400x400x1x3)

# axes3d

## Arguments:

- `_size_x,_size_y,_size_z,_font_size>0,_label_x,_label_y,_label_z,_is_origin={ 0:no | 1:yes }`

## Description:

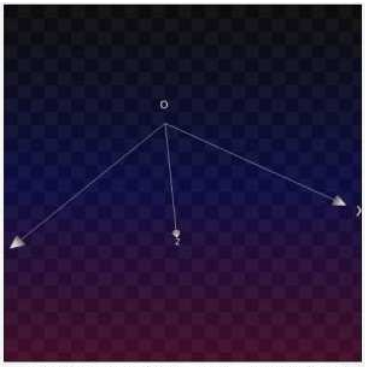
Input 3D axes with specified sizes along the x,y and z orientations.

## Default values:

`size_x=size_y=size_z=1`, `font_size=23`, `label_x=X`, `label_y=Y`, `label_z=Z` and `is_origin=1`

## Example of use:

```
axes3d ,
```



[0]: '[3d axes]' (64 vert., 103 prim.)

# balance_gamma

## Arguments:

- `_ref_color1,...`

## Description:

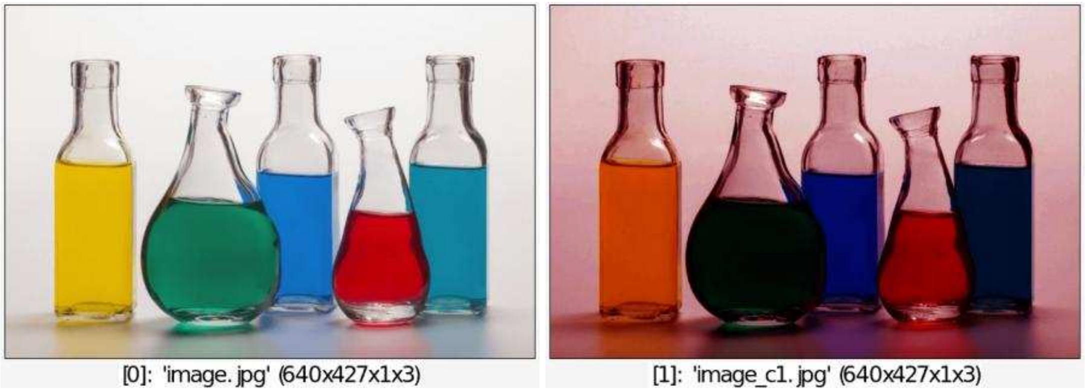
Compute gamma-corrected color balance of selected image, with respect to specified reference color.

## Default values:

`ref_color1=128`.

## Example of use:

```
image.jpg +balance_gamma 128,64,64
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

# ball

## Arguments:

- `_size>0, _R,_G,_B,0<=_specular_light<=8,0<=_specular_size<=8,_shadow>=0`

## Description:

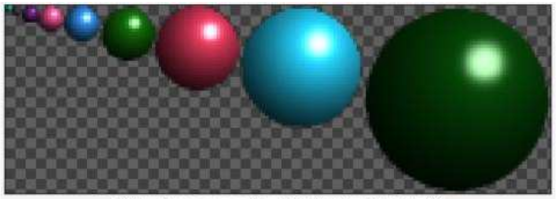
Input a 2D RGBA colored ball sprite.

## Default values:

`size=64`, `R=255`, `G=R`, `B=R`, `specular_light=0.8`, `specular_size=1` and `shading=1.5`.

## Example of use:

```
repeat 9 { ball {1.5^($>+2)},${-rgb} } append x
```



[0]: '[unnamed]_c1' (168x58x1x4)

# bandpass

## Arguments:

- `_min_freq[%],_max_freq[%]`

## Description:

Apply bandpass filter to selected images.

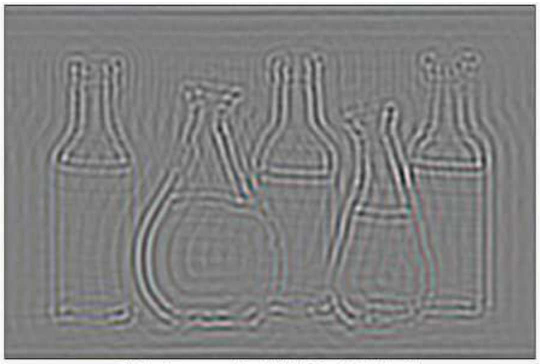
## Default values:

`min_freq=0` and `max_freq=20%`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg bandpass 1%,3%
```



[0]: 'image.jpg' (640x427x1x3)

# barycenter

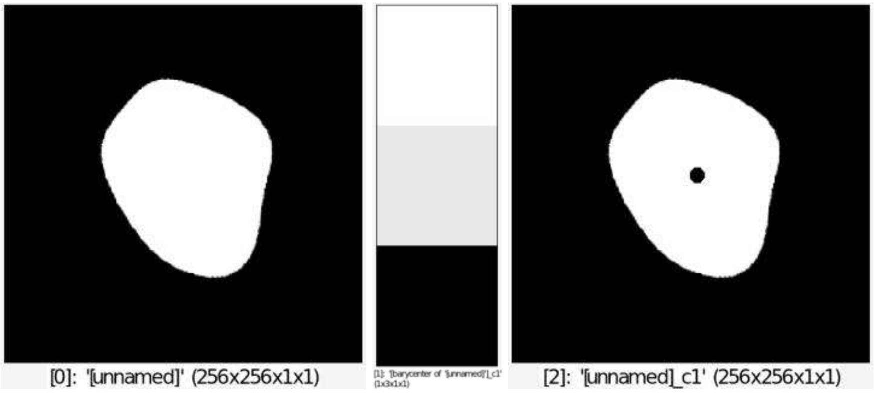
**No arguments**

## Description:

Compute the barycenter vector of pixel values.

## Example of use:

```
256,256 ellipse 50%,50%,20%,20%,0,1,1 deform 20 +barycenter
+ellipse[-2] {@0,1},5,5,0,10
```


[0]: '[unnamed]' (256x256x1x1)
[1]: '[barycenter of '[unnamed]']_c1' (1x3x1x1)
[2]: '[unnamed]_c1' (256x256x1x1)

# base642img

## Arguments:

- `"base64_string"`

## Description:

Decode given base64-encoded string as a newly inserted image at the end of the list.

The argument string must have been generated using command `img2base64`.

# base642uint8

## Arguments:

- `"base64_string"`

## Description:

Decode given base64-encoded string as a newly inserted 1-column image at the end of the list.

The argument string must have been generated using command `uint82base64`.

# basename

## Arguments:

- `file_path,_variable_name_for_folder`

### Description:

Return the basename of a file path, and opt. its folder location.

When specified `variable_name_for_folder` must starts by an underscore
(global variable accessible from calling function).

## bayer2rgb

### Arguments:

- `_GM_smoothness,_RB_smoothness1,_RB_smoothness2`

### Description:

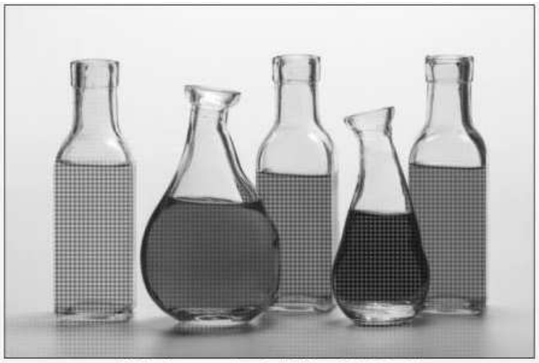
Transform selected RGB-Bayer sampled images to color images.

### Default values:

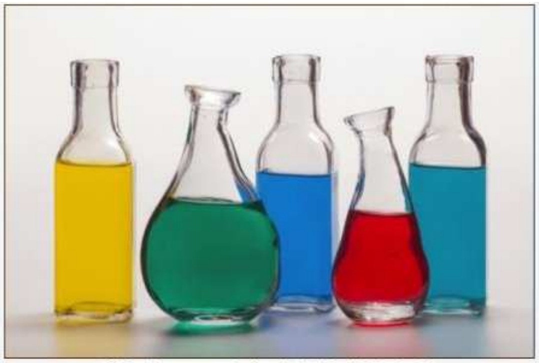
`GM_smoothness=RB_smoothness=1` and `RB_smoothness2=0.5`.

### Example of use:

```
image.jpg rgb2bayer 0 +bayer2rgb 1,1,0.5
```



[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x3)

## betti

#### No arguments

### Description:

Compute Betti numbers B0,B1 and B2 from selected 3D binary shapes.

Values B0,B1 and B2 are returned in the status. When multiple images are selected, the B0,B1,B2 of each image are concatenated in the status.
(see `https://en.wikipedia.org/wiki/Betti_number` for details about Betti numbers).

# bilateral

Built-in command

## Arguments:

- `[guide],std_deviation_s[%]>=0,std_deviation_r[%]>=0,_sampling_s>=0,_sampling_`
  or
- `std_deviation_s[%]>=0,std_deviation_r[%]>=0,_sampling_s>=0,_sampling_r>=0`

## Description:

Blur selected images by anisotropic (eventually joint/cross) bilateral filtering.

If a guide image is provided, it is used for drive the smoothing filter.
A guide image must be of the same xyz-size as the selected images.
Set `sampling` arguments to `0` for automatic adjustment.

## Example of use:

```
image.jpg repeat 5 { bilateral 10,10 }
```

[0]: 'image.jpg' (640x427x1x3)

# bin

## Arguments:

- `binary_int1,...`

## Description:

Print specified binary integers into their octal, decimal, hexadecimal and string representations.

# bin2dec

## Arguments:

- `binary_int1,...`

## Description:

Convert specified binary integers into their decimal representations.

---

# blend

## Arguments:

- `[layer],blending_mode,_opacity[%],_selection_is={ 0:base-layers | 1:top-layers }` or
- `blending_mode,_opacity[%]`

## Description:

Blend selected G,GA,RGB or RGBA images by specified layer or blend all selected images together,

using specified blending mode.
`blending_mode` can be { add | alpha | and | average | blue | burn | darken | difference |
divide | dodge | edges | exclusion | freeze | grainextract | grainmerge | green | hardlight |
hardmix | hue | interpolation | lchlightness | lighten | lightness | linearburn | linearlight |
luminance |
multiply | negation | or | overlay | pinlight | red | reflect | saturation |
screen | seamless | seamless_mixed | shapeareamax | shapeareamax0 | shapeareamin |
shapeareamin0 |
shapeaverage | shapeaverage0 | shapemedian | shapemedian0 | shapemin | shapemin0 |
shapemax | shapemax0 |
shapeprevalent | softburn | softdodge | softlight | stamp | subtract | value | vividlight | xor }.
`opacity` must be in range `[0,1]` (or `[0%,100%]`).

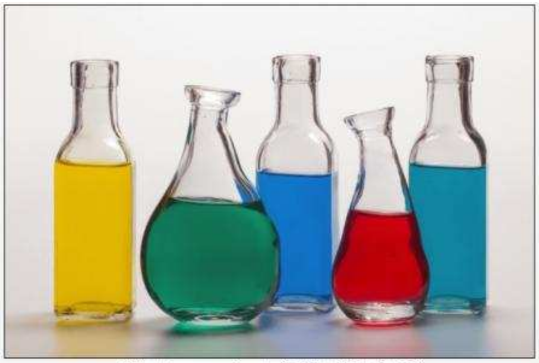
## Default values:

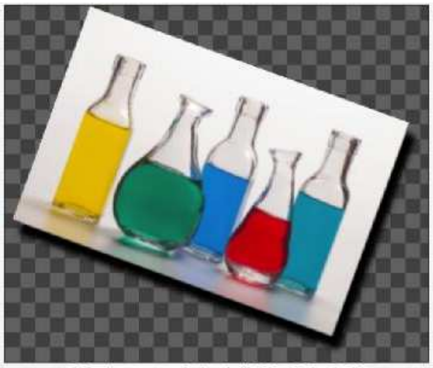
`blending_mode=alpha`, `opacity=1` and `selection_is=0`.

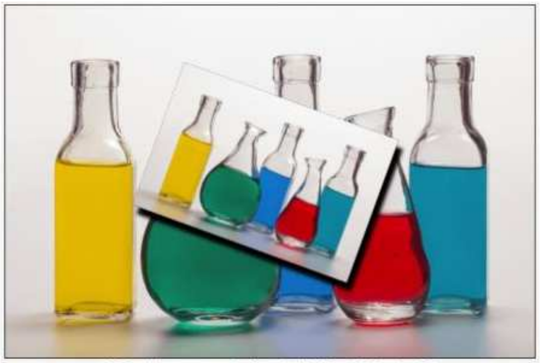
## Examples of use:

- **Example #1**

```
image.jpg +drop_shadow , rescale2d[-1] ,200 rotate[-1] 20 +blend alpha display_rgba[-2]
```

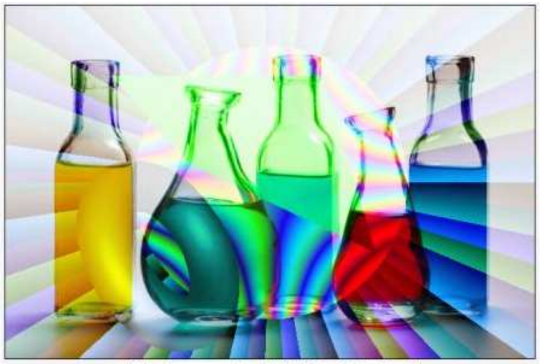

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (343x288x1x3)


[2]: 'image_c1.jpg' (640x427x1x3)

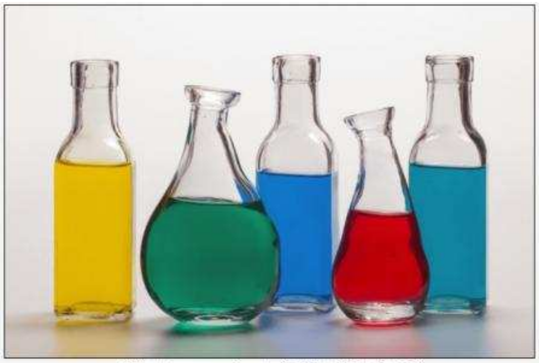
- **Example #2**

```
image.jpg testimage2d {w},{h} blend overlay
```

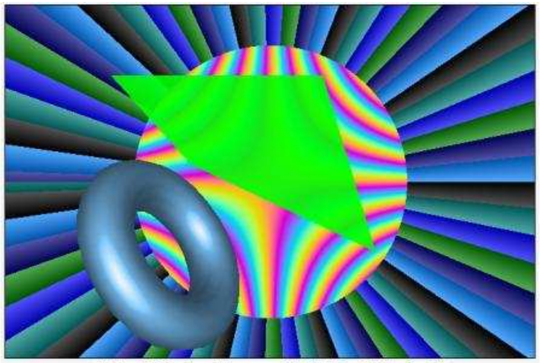

[0]: 'image.jpg' (640x427x1x3)

- **Example #3**

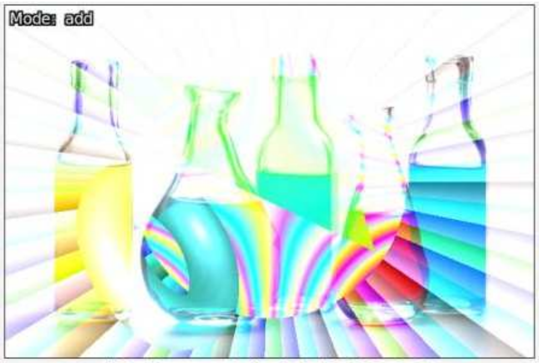
```
command "ex : $""=arg repeat $""# +blend[0,1] ${arg{$>+1}}
text_outline[-1] Mode:\" \"${arg{$>+1}},2,2,23,2,1,255 done"
image.jpg testimage2d {w},{h} ex
add,alpha,and,average,blue,burn,darken
```



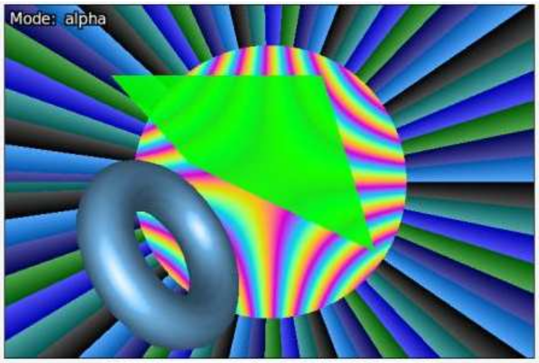
[0]: 'image.jpg' (640x427x1x3)



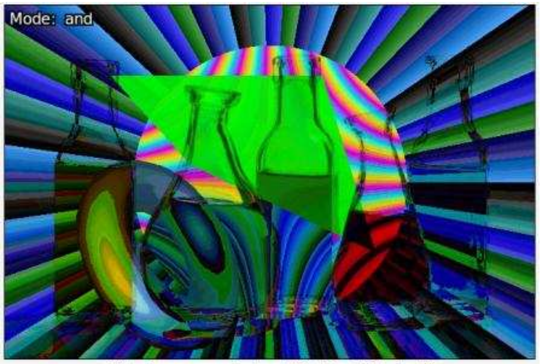
[1]: '[2D test image]' (640x427x1x3)



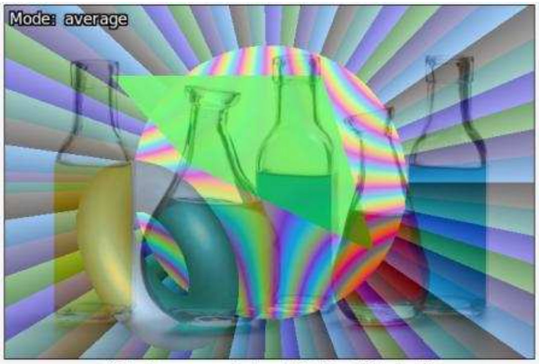
[2]: 'image_c1.jpg' (640x427x1x3)



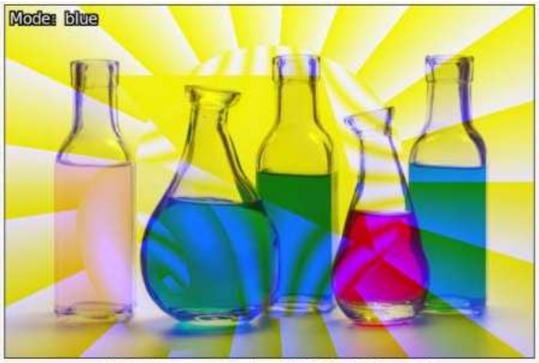
[3]: 'image_c1.jpg' (640x427x1x3)



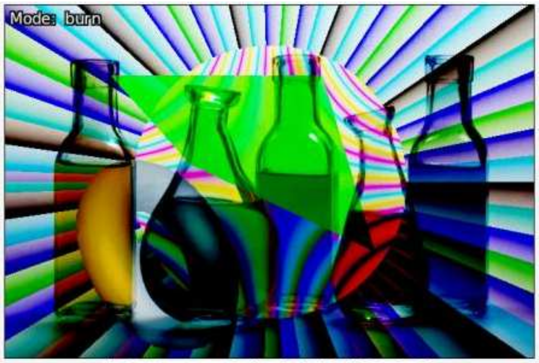
[4]: 'image_c1.jpg' (640x427x1x3)



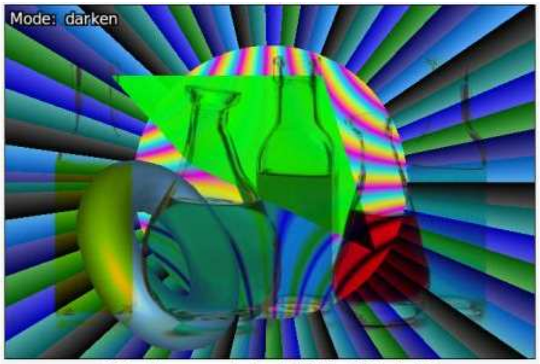
[5]: 'image_c1.jpg' (640x427x1x3)



[6]: 'image_c1.jpg' (640x427x1x3)



[7]: 'image_c1.jpg' (640x427x1x3)

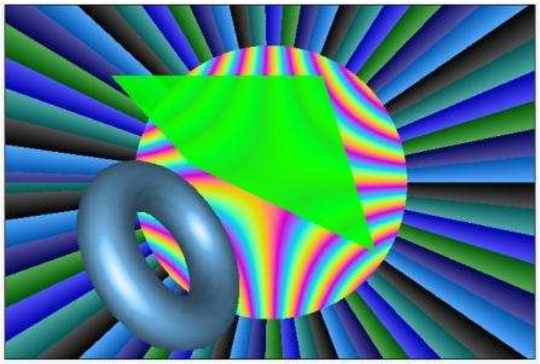


[8]: 'image_c1.jpg' (640x427x1x3)

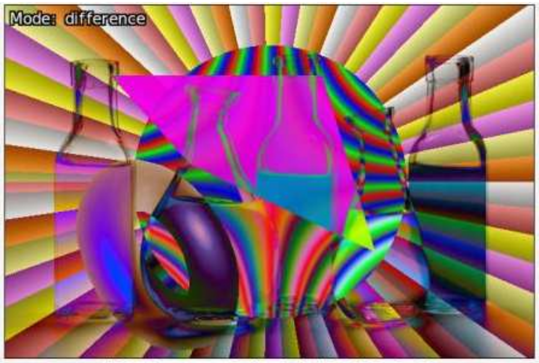
- **Example #4**

```
command "ex : $""=arg repeat $""# +blend[0,1] ${arg{$>+1}}
text_outline[-1] Mode:\" \"${arg{$>+1}},2,2,23,2,1,255 done"
image.jpg testimage2d {w},{h} ex
difference,divide,dodge,exclusion,freeze,grainextract,grainmerge
```

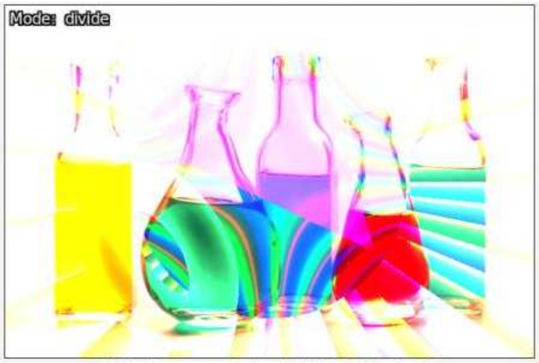
[0]: 'image.jpg' (640x427x1x3)



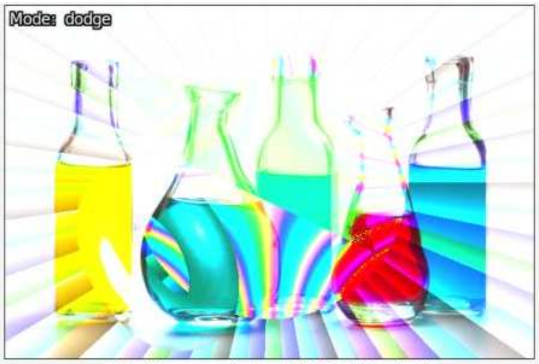
[1]: '[2D test image]' (640x427x1x3)



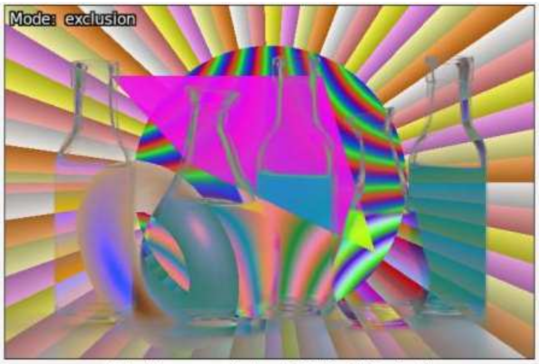
[2]: 'image_c1.jpg' (640x427x1x3)



[3]: 'image_c1.jpg' (640x427x1x3)



[4]: 'image_c1.jpg' (640x427x1x3)



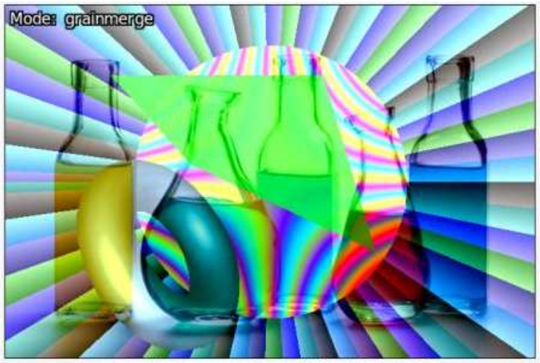
[5]: 'image_c1.jpg' (640x427x1x3)

Mode: freeze

[6]: 'image_c1.jpg' (640x427x1x3)

Mode: grainextract

[7]: 'image_c1.jpg' (640x427x1x3)

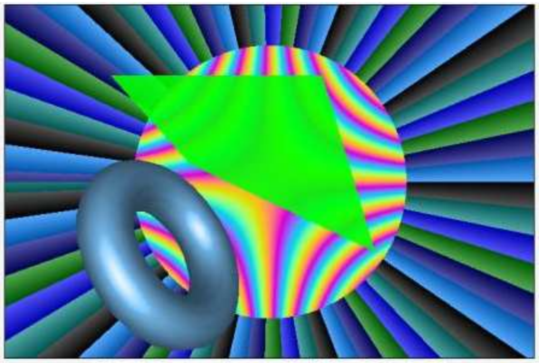


[8]: 'image_c1.jpg' (640x427x1x3)

- **Example #5**

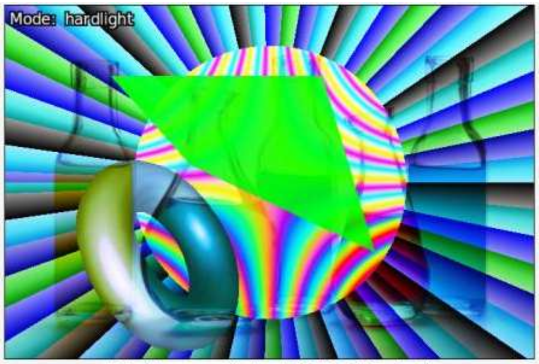
```
command "ex : $""=arg repeat $""# +blend[0,1] ${arg{$>+1}}
text_outline[-1] Mode:\" \"${arg{$>+1}},2,2,23,2,1,255 done"
image.jpg testimage2d {w},{h} ex
green,hardlight,hardmix,hue,interpolation,lighten,lightness
```

[0]: 'image.jpg' (640x427x1x3)

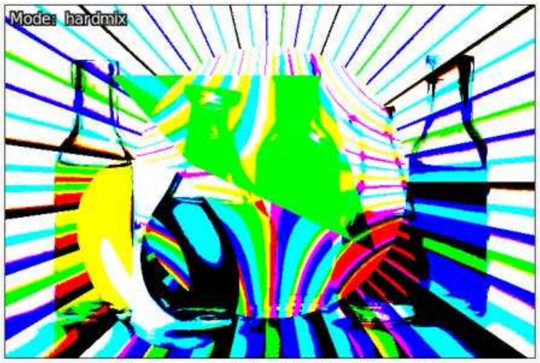


[1]: '[2D test image]' (640x427x1x3)

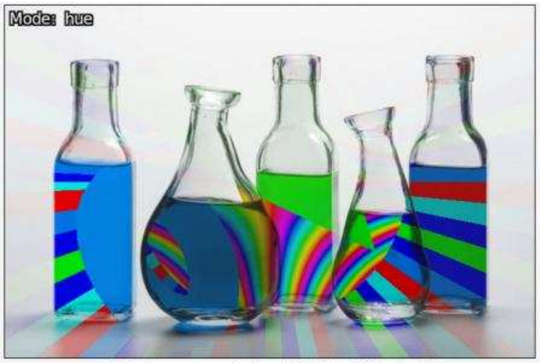
[2]: 'image_c1.jpg' (640x427x1x3)



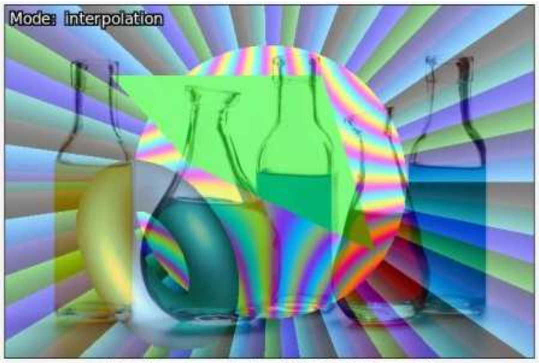
[3]: 'image_c1.jpg' (640x427x1x3)



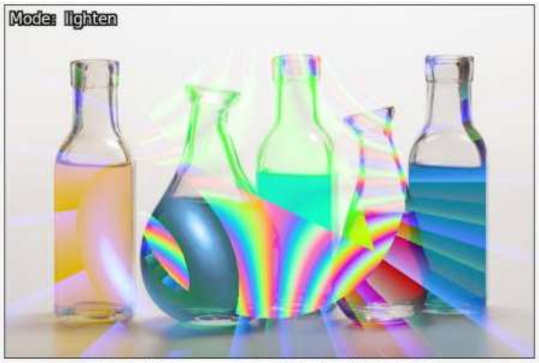
[4]: 'image_c1.jpg' (640x427x1x3)



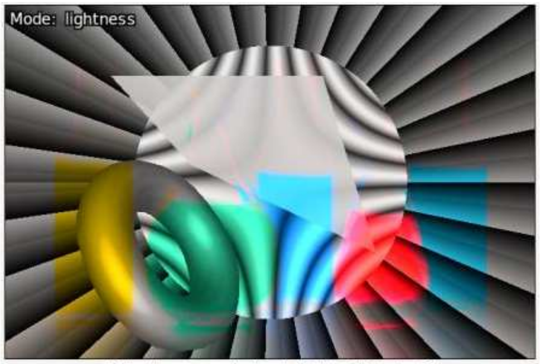
[5]: 'image_c1.jpg' (640x427x1x3)



[6]: 'image_c1.jpg' (640x427x1x3)



[7]: 'image_c1.jpg' (640x427x1x3)

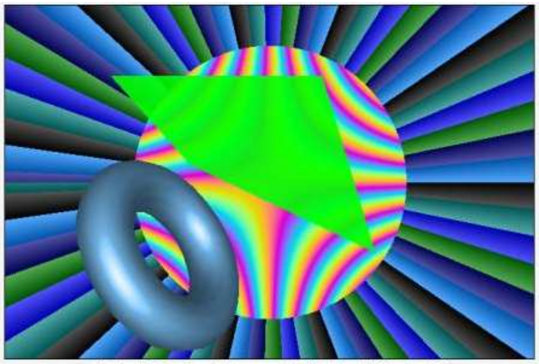


[8]: 'image_c1.jpg' (640x427x1x3)

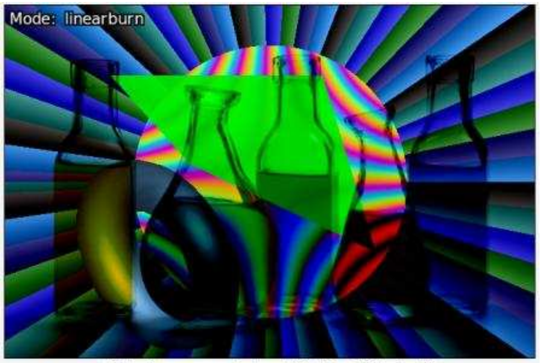
- **Example #6**

```
command "ex : $""=arg repeat $""# +blend[0,1] ${arg{$>+1}}
text_outline[-1] Mode:\" \"${arg{$>+1}},2,2,23,2,1,255 done"
image.jpg testimage2d {w},{h} ex
linearburn,linearlight,luminance,multiply,negation,or,overlay
```

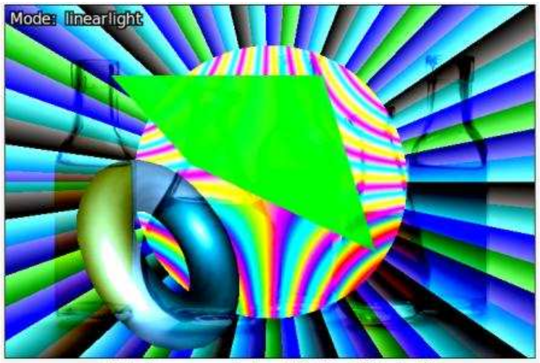
[0]: 'image.jpg' (640x427x1x3)



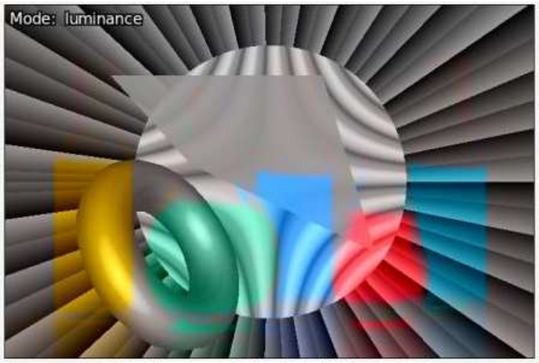
[1]: '[2D test image]' (640x427x1x3)



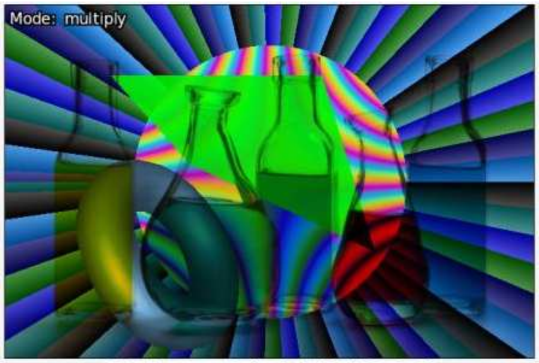
[2]: 'image_c1. jpg' (640x427x1x3)



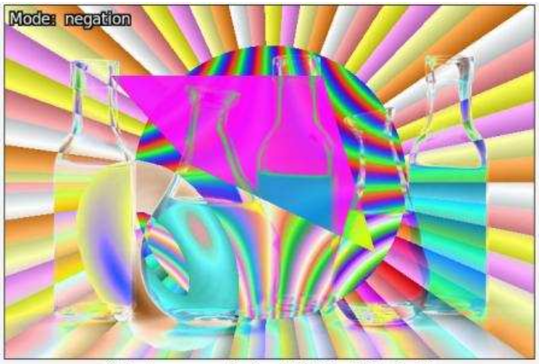
[3]: 'image_c1. jpg' (640x427x1x3)



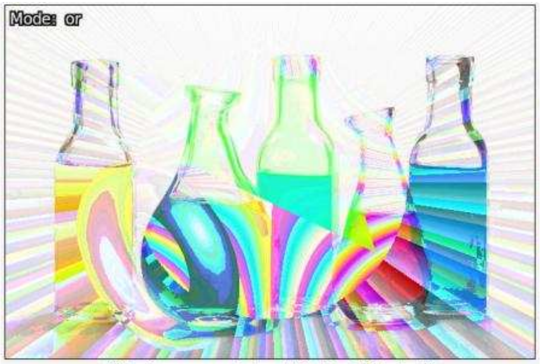
[4]: 'image_c1. jpg' (640x427x1x3)



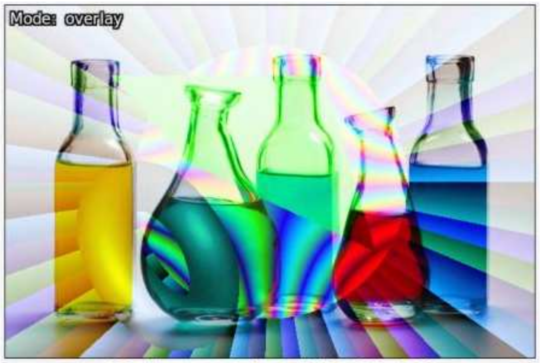
[5]: 'image_c1. jpg' (640x427x1x3)



[6]: 'image_c1. jpg' (640x427x1x3)



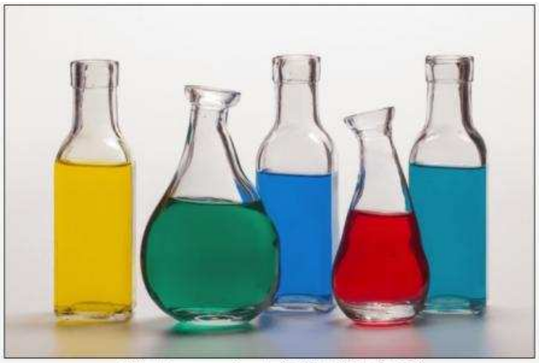
[7]: 'image_c1. jpg' (640x427x1x3)



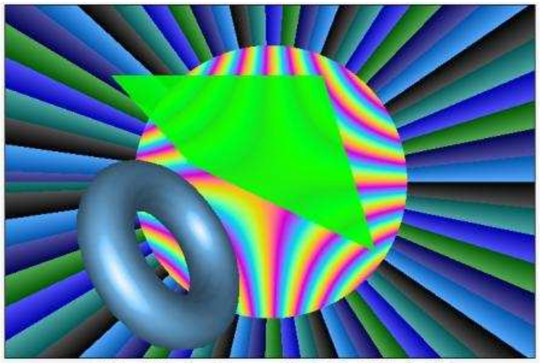
[8]: 'image_c1. jpg' (640x427x1x3)

- **Example #7**

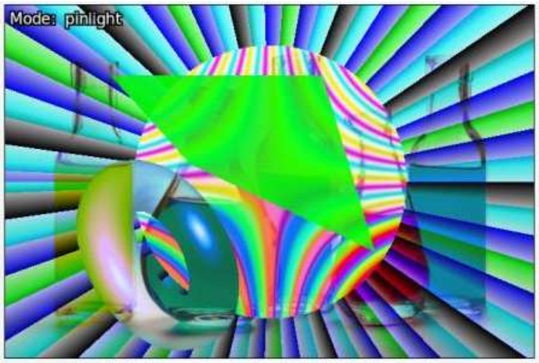
```
command "ex : $""=arg repeat $""# +blend[0,1] ${arg{$>+1}}
text_outline[-1] Mode:\" \"${arg{$>+1}},2,2,23,2,1,255 done"
image.jpg testimage2d {w},{h} ex
pinlight,red,reflect,saturation,screen,shapeaverage,softburn
```



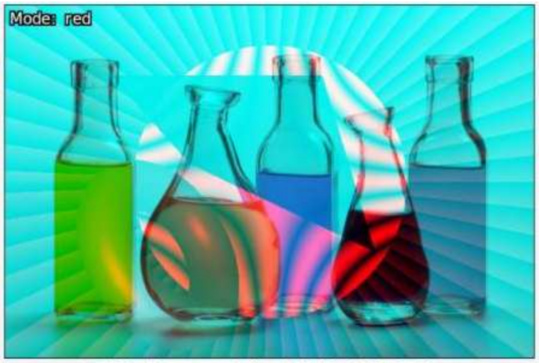
[0]: 'image.jpg' (640x427x1x3)



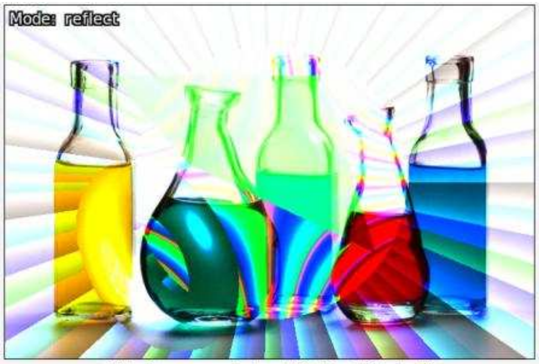
[1]: '[2D test image]' (640x427x1x3)



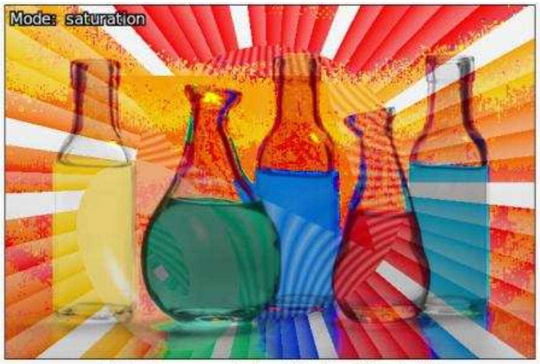
[2]: 'image_c1.jpg' (640x427x1x3)



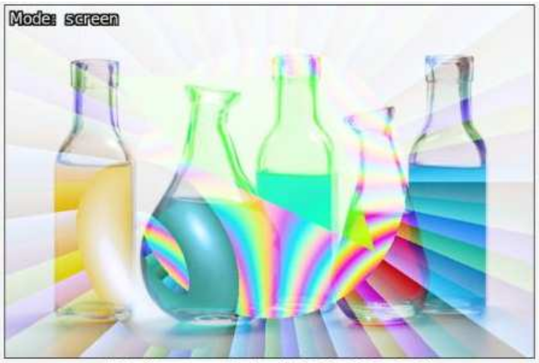
[3]: 'image_c1.jpg' (640x427x1x3)



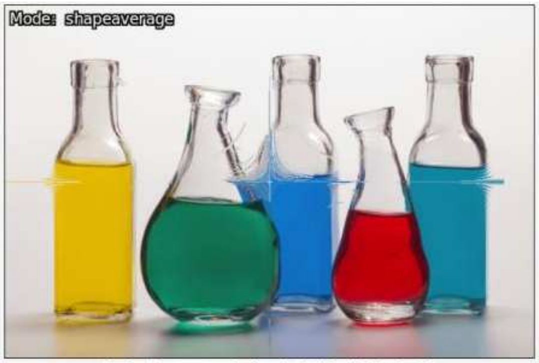
[4]: 'image_c1.jpg' (640x427x1x3)



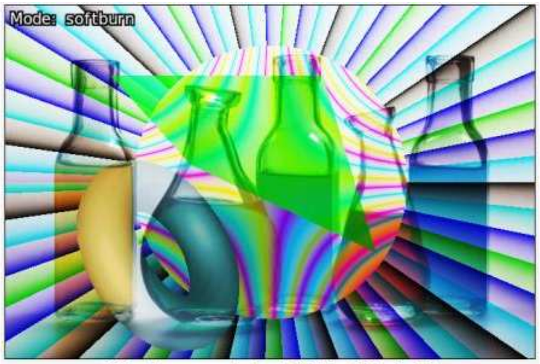
[5]: 'image_c1.jpg' (640x427x1x3)



[6]: 'image_c1.jpg' (640x427x1x3)



[7]: 'image_c1.jpg' (640x427x1x3)

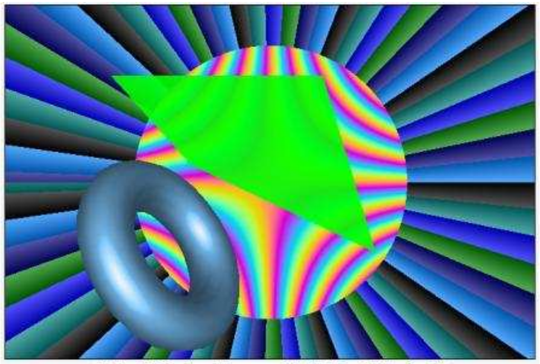


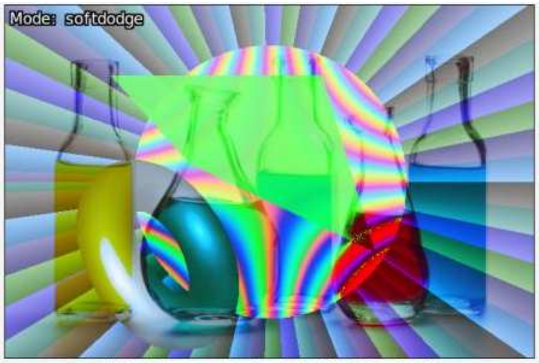
[8]: 'image_c1.jpg' (640x427x1x3)

- **Example #8**

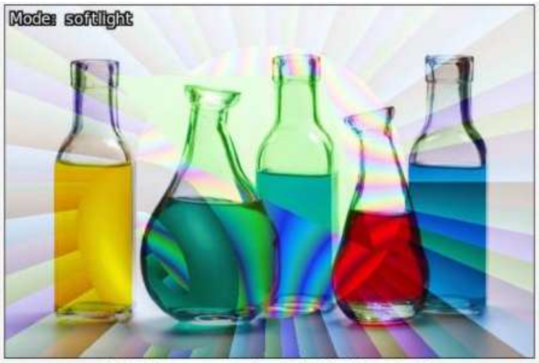
```
command "ex : $""=arg repeat $""# +blend[0,1] ${arg{$>+1}}
text_outline[-1] Mode:\" \"${arg{$>+1}},2,2,23,2,1,255 done"
image.jpg testimage2d {w},{h} ex
softdodge,softlight,stamp,subtract,value,vividlight,xor
```

[0]: 'image.jpg' (640x427x1x3)

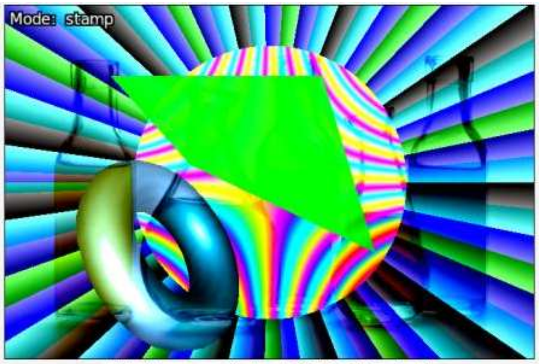

[1]: '[2D test image]' (640x427x1x3)



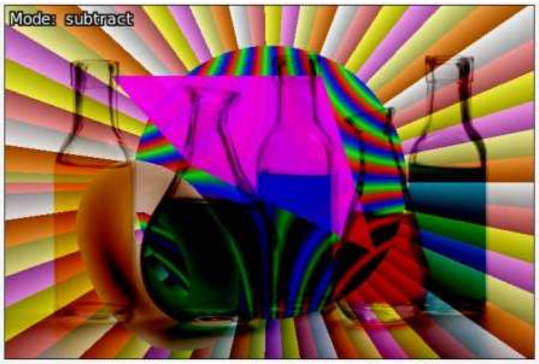
[2]: 'image_c1.jpg' (640x427x1x3)



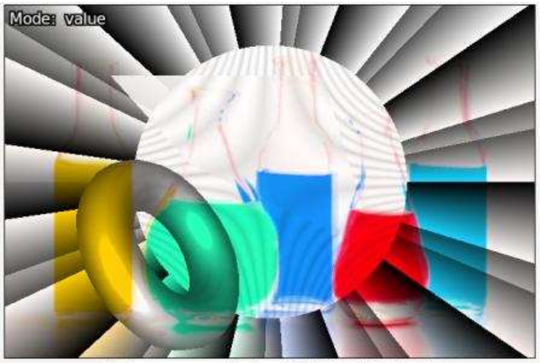
[3]: 'image_c1.jpg' (640x427x1x3)



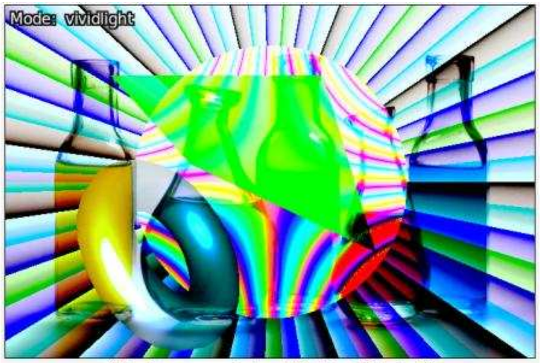
[4]: 'image_c1.jpg' (640x427x1x3)



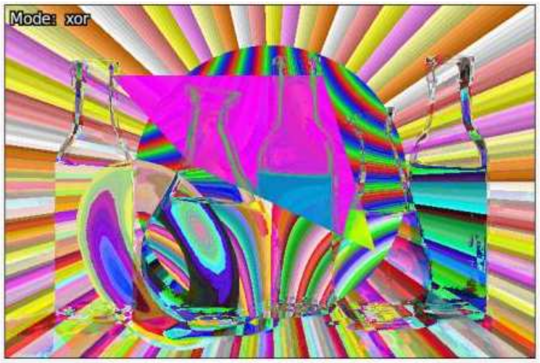
[5]: 'image_c1.jpg' (640x427x1x3)



[6]: 'image_c1.jpg' (640x427x1x3)



[7]: 'image_c1.jpg' (640x427x1x3)



[8]: 'image_c1.jpg' (640x427x1x3)

---

# blend_edges

## Arguments:

- `smoothness[%]>=0`

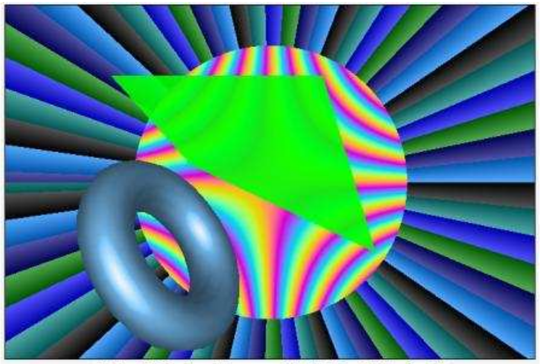
## Description:

Blend selected images togethers using `edges` mode.

## Example of use:

```
image.jpg testimage2d {w},{h} +blend_edges 0.8
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[2D test image]' (640x427x1x3)

[2]: 'image_c1.jpg' (640x427x1x3)

# blend_fade

## Arguments:

- `[fading_shape]`

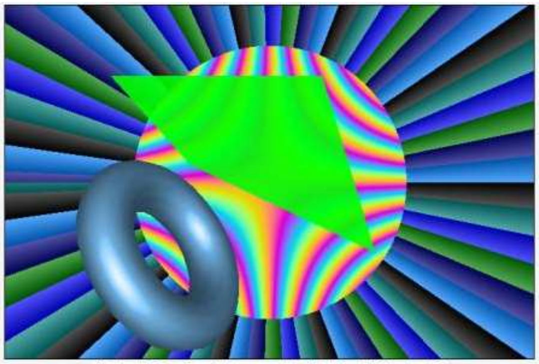
## Description:

Blend selected images together using specified fading shape.

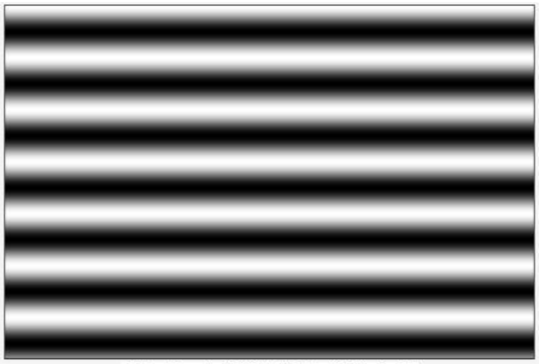
## Example of use:

```
image.jpg testimage2d {w},{h} 100%,100%,1,1,'cos(y/10)' normalize[-1]
0,1 +blend_fade[0,1] [2]
```

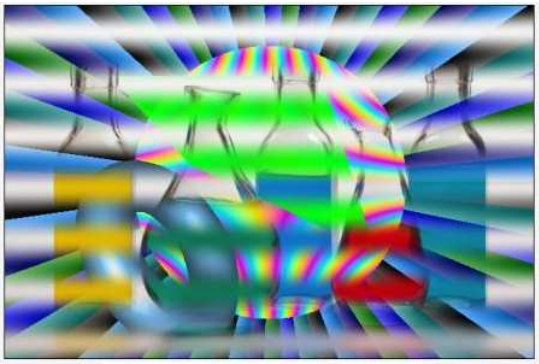
[0]: 'image.jpg' (640x427x1x3)



[1]: '[2D test image]' (640x427x1x3)



[2]: '[cos(y/10)]' (640x427x1x1)



[3]: 'image_c1.jpg' (640x427x1x3)

# blend_median

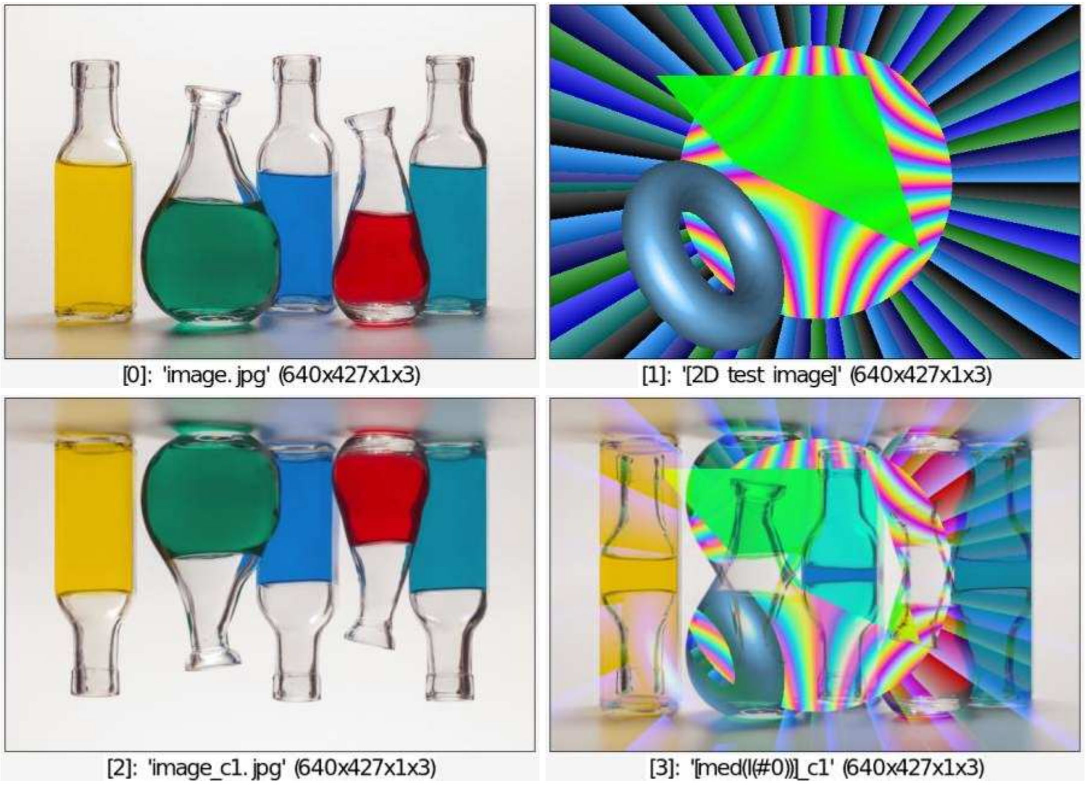
**No arguments**

## Description:

Blend selected images together using `median` mode.

## Example of use:

```
image.jpg testimage2d {w},{h} +mirror[0] y +blend_median
```


[0]: 'image.jpg' (640x427x1x3)
[1]: '[2D test image]' (640x427x1x3)
[2]: 'image_c1.jpg' (640x427x1x3)
[3]: '[med(I(#0))]_c1' (640x427x1x3)

# blend_seamless

## Arguments:

- `_is_mixed_mode={ 0 | 1 },_inner_fading[%]>=0,_outer_fading[%]>=0`

## Description:

Blend selected images using a seamless blending mode (Poisson-based).

## Default values:

`is_mixed=0`, `inner_fading=0` and `outer_fading=100%`.

# blur

Built-in command

## Arguments:

- `std_deviation>=0[%],_boundary_conditions,_kernel` or
- `axes,std_deviation>=0[%],_boundary_conditions,_kernel`

## Description:

Blur selected images by a deriche or gaussian filter (recursive implementation).

(*equivalent to shortcut command* `b`).

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`kernel` can be ***{ 0:deriche | 1:gaussian }***.
When specified, argument `axes` is a sequence of ***{ x | y | z | c }***.
Specifying one axis multiple times apply also the blur multiple times.

## Default values:

`boundary_conditions=1` and `kernel=1`.

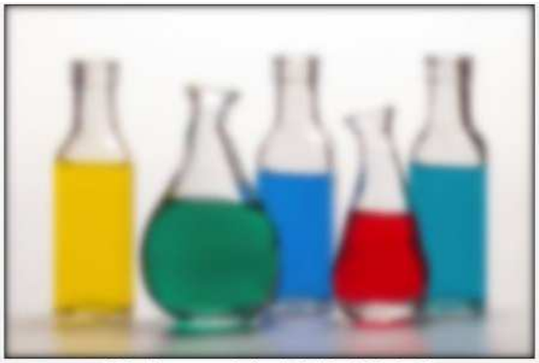
This command has a **tutorial page**.

## Examples of use:

### • Example #1

```
image.jpg +blur 5,0 +blur[0] 5,1
```

[0]: 'image.jpg' (640x427x1x3)

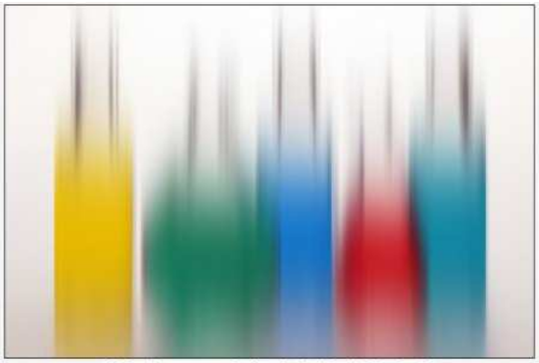

[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg +blur y,10%
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# blur_angular

## Arguments:

- `amplitude[%],_center_x[%],_center_y[%]`

## Description:

Apply angular blur on selected images.

## Default values:

`center_x=center_y=50%`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg blur_angular 2%
```

[0]: 'image.jpg' (640x427x1x3)

# blur_bloom

## Arguments:

- `_amplitude>=0,_ratio>=0,_nb_iter>=0,_blend_operator={ + | max | min },_kernel={ 0:deriche | 1:gaussian | 2:box | 3:triangle | 4:quadratic },_normalize_scales={ 0 | 1 },_axes`

## Description:

Apply a bloom filter that blend multiple blur filters of different radii,

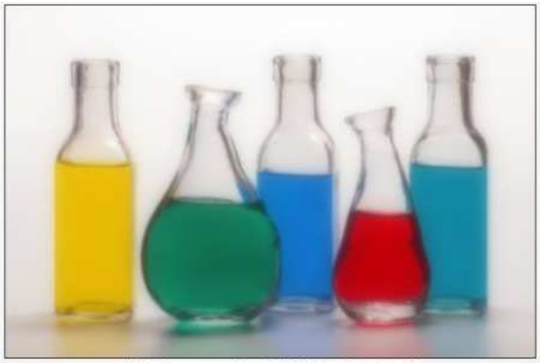
resulting in a larger but sharper glare than a simple blur.
When specified, argument `axes` is a sequence of *{ x | y | z | c }*.
Specifying one axis multiple times apply also the blur multiple times.
Reference: Masaki Kawase, "Practical Implementation of High Dynamic Range Rendering", GDC 2004.

## Default values:

`amplitude=1`, `ratio=2`, `nb_iter=5`, `blend_operator=+`, `kernel=1`, `normalize_scales=0` and `axes=(all)`

## Example of use:

```
image.jpg blur_bloom ,
```



[0]: 'image.jpg' (640x427x1x3)

# blur_linear

## Arguments:

- `amplitude1[%],_amplitude2[%],_angle,_boundary_conditions={ 0:dirichlet | 1:neumann }`

## Description:

Apply linear blur on selected images, with specified angle and amplitudes.

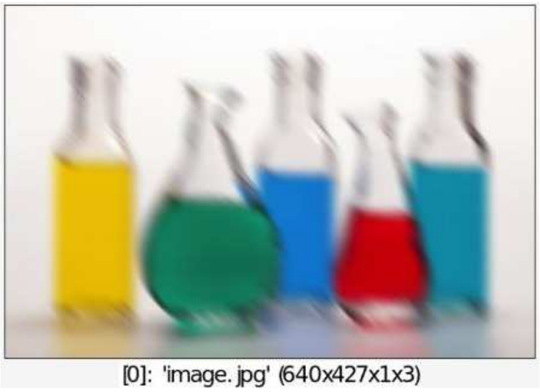
## Default values:

`amplitude2=0`, `angle=0` and `boundary_conditions=1`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg blur_linear 10,0,45
```



[0]: 'image.jpg' (640x427x1x3)

# blur_radial

## Arguments:

- `amplitude[%],_center_x[%],_center_y[%]`

## Description:

Apply radial blur on selected images.

## Default values:

`center_x=center_y=50%`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg blur_radial 2%
```

[0]: 'image.jpg' (640x427x1x3)

# blur_selective

## Arguments:

- `sigma>=0,_edges>0,_nb_scales>0`

## Description:

Blur selected images using selective gaussian scales.

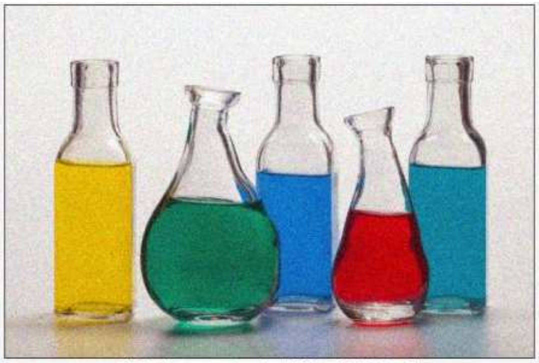
## Default values:

`sigma=5`, `edges=0.5` and `nb_scales=5`.

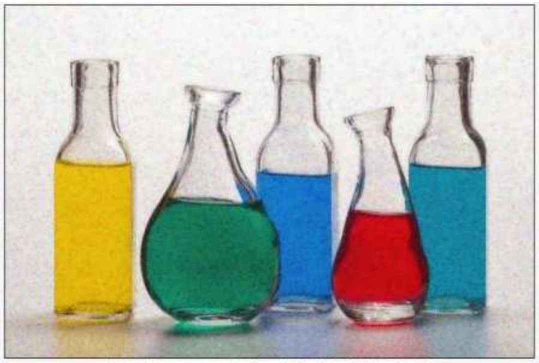
This command has a **tutorial page**.

## Example of use:

```
image.jpg noise 20 cut 0,255 +local[-1] repeat 4 { blur_selective , }
done
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# blur_x

## Arguments:

- `amplitude[%]>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Blur selected images along the x-axis.

## Default values:

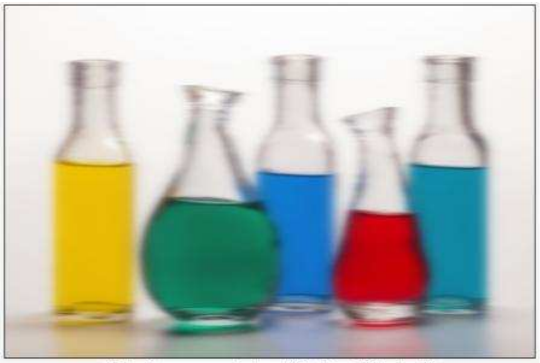
`boundary_conditions=1`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg +blur_x 6
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# blur_xy

## Arguments:

- `amplitude_x[%],amplitude_y[%],_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Blur selected images along the X and Y axes.

## Default values:

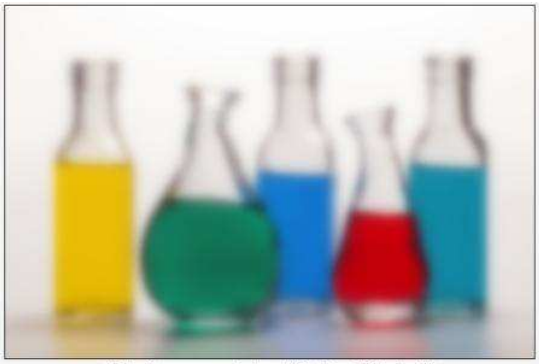
`boundary_conditions=1`.

This command has a **tutorial page**.

### Example of use:

```
image.jpg +blur_xy 6
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

## blur_xyz

### Arguments:

- `amplitude_x[%],amplitude_y[%],amplitude_z,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

### Description:

Blur selected images along the X, Y and Z axes.

### Default values:

`boundary_conditions=1`.

This command has a **tutorial page**.

## blur_y

### Arguments:

- `amplitude[%]>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

### Description:

Blur selected images along the y-axis.

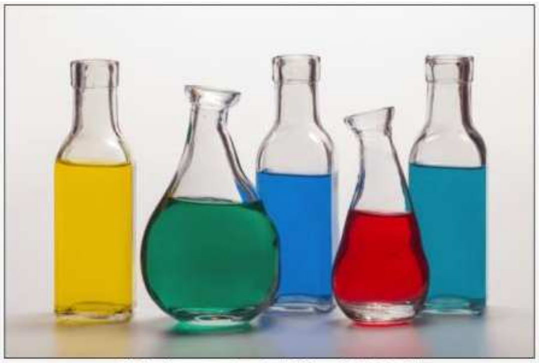
### Default values:

`boundary_conditions=1`.

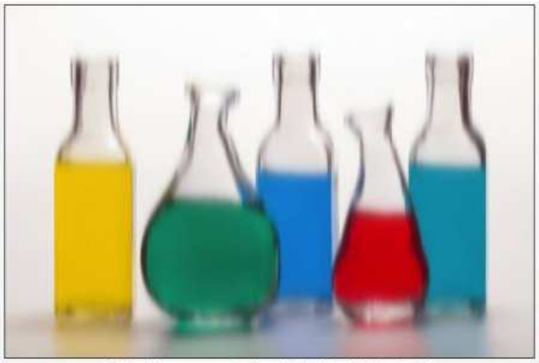
This command has a **tutorial page**.

## Example of use:

```
image.jpg +blur_y 6
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# blur_z

## Arguments:

- `amplitude[%]>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Blur selected images along the z-axis.

## Default values:

`boundary_conditions=1`.

This command has a **tutorial page**.

# boundingbox3d

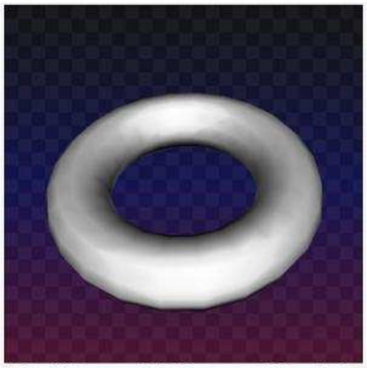
### No arguments

## Description:

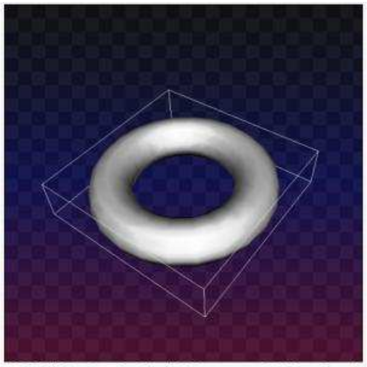
Replace selected 3D objects by their 3D bounding boxes.

## Example of use:

```
torus3d 100,30 +boundingbox3d +3d[-1] [-2]
```



[0]: '[3D torus]' (288 vert., 288 prim.)



[1]: '[3D box]_c1' (296 vert., 300 prim.)

# box3d

## Arguments:

- `_size_x,_size_y,_size_z`

## Description:

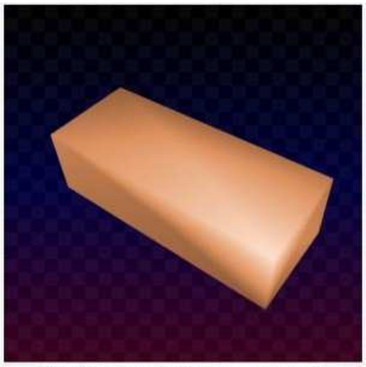
Input 3D box at (0,0,0), with specified geometry.

## Default values:

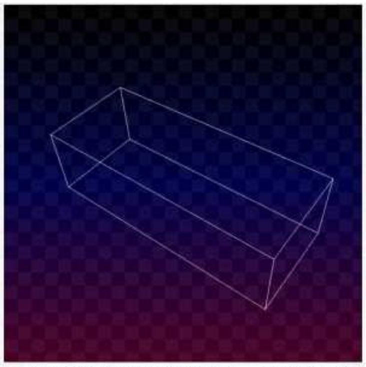
`size_x=1` and `size_z=size_y=size_x`.

## Example of use:

```
box3d 100,40,30 +primitives3d 1 color3d[-2] ${-rgb}
```



[0]: '[3D box]' (8 vert., 6 prim.)



[1]: '[3D box]_c1' (8 vert., 12 prim.)

# boxfilter

Built-in command

## Arguments:

- `size>=0[%],_order,_boundary_conditions,_nb_iter>=0` or

- `axes,size>=0[%],_order,_boundary_conditions,_nb_iter>=0`

## Description:

Blur selected images by a box filter of specified size (fast recursive implementation).

`order` can be ***{ 0:smooth | 1:1st-derivative | 2:2nd-derivative }***.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
When specified, argument `axes` is a sequence of ***{ x | y | z | c }***.
Specifying one axis multiple times apply also the blur multiple times.

## Default values:

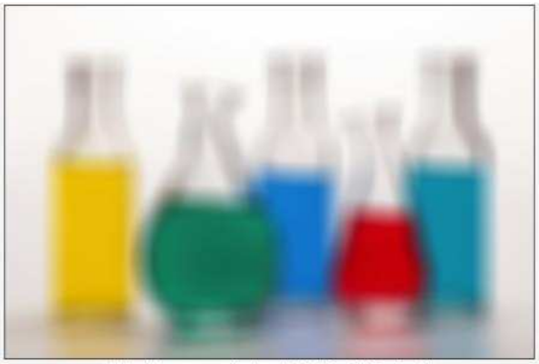
`order=0`, `boundary_conditions=1` and `nb_iter=1`.

## Examples of use:

### • Example #1

```
image.jpg +boxfilter 5%
```

[0]: 'image.jpg' (640x427x1x3)

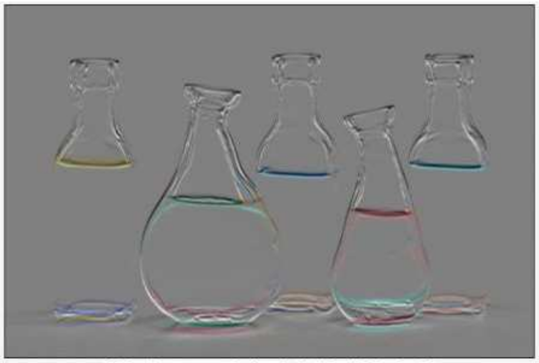


[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +boxfilter y,3,1
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## boxfitting

### Arguments:

- `_min_box_size>=1,_max_box_size>=0,_initial_density>=0,_min_spacing>0`

### Description:

Apply box fitting effect on selected images, as displayed the web page:

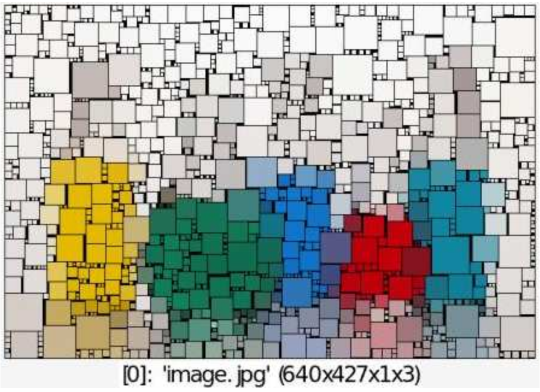
**http://www.complexification.net/gallery/machines/boxFittingImg/**.

### Default values:

`min_box_size=1`, `max_box_size=0`, `initial_density=0.25` and `min_spacing=1`.

### Example of use:

```
image.jpg boxfitting ,
```



[0]: 'image.jpg' (640x427x1x3)

---

## break

**Built-in command**

**No arguments**

### Description:

Break current `do...while`, `for...done`, `foreach...done`, `local...done` or `repeat...done` block.

### Example of use:

```
image.jpg repeat 10 blur 1 if 1==1 break fi deform 10 done
```

[0]: 'image.jpg' (640x427x1x3)

# brushify

## Arguments:

- `[brush],_brush_nb_sizes>=1,0<=_brush_min_size_factor<=1,_brush_nb_orientation`
  `0<=_brush_light_strength<=1,_brush_opacity,_painting_density[%]>=0,0<=_painti`
  `0<=_painting_angle_dispersion<=1`

## Description:

Apply specified brush to create painterly versions of specified images.

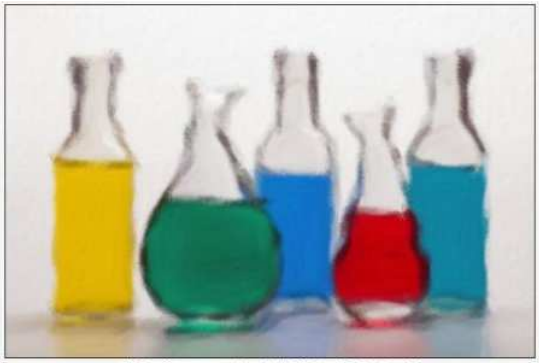
`brush_light_type` can be *{ 0:none | 1:flat | 2:darken | 3:lighten | 4:full }*.

## Default values:

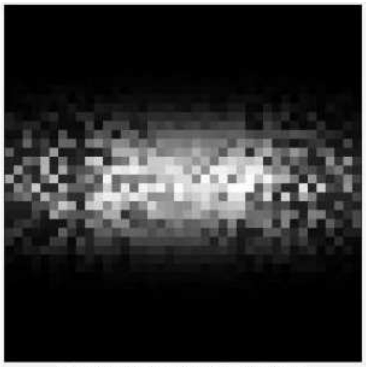
`brush_nb_sizes=3`, `brush_min_size_factor=0.66`, `brush_nb_orientations=12`, `brush_light_type=0`, `brush_light_strength=0.25`, `brush_opacity=0.8`, `painting_density=20%`, `painting_contours_coherence=0.9`, `painting_orientation_coherence=0.9`, `painting_coherence_alpha=1`, `painting_coherence_sigma=1`, `painting_primary_angle=0`, `painting_angle_dispersion=0.2`

## Example of use:

```
image.jpg 40,40 gaussian[-1] 10,4 spread[-1] 10,0 brushify[0] [1],1
```



[0]: 'image.jpg' (640x427x1x4)



[1]: '[unnamed]' (40x40x1x1)

# bsl

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

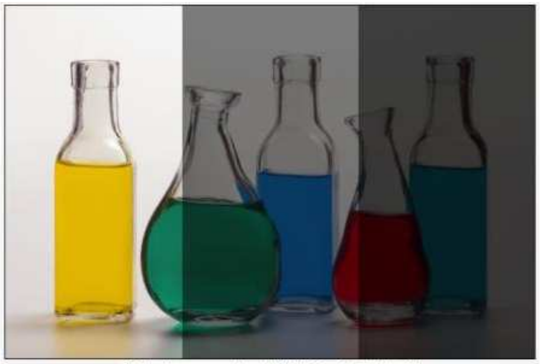
## Description:

Compute the bitwise left shift of selected images with specified value, image or mathematical expression, or compute the pointwise sequential bitwise left shift of selected images.

(*equivalent to shortcut command* `<<`).

## Example of use:

```
image.jpg bsl 'round(3*x/w,0)' cut 0,255
```



[0]: 'image.jpg' (640x427x1x3)

# bsr

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

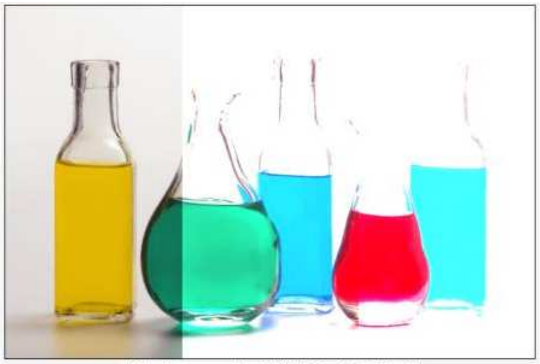
## Description:

Compute the bitwise right shift of selected images with specified value, image or mathematical expression, or compute the pointwise sequential bitwise right shift of selected images.

(*equivalent to shortcut command* `>>`).

### Example of use:

```
image.jpg bsr 'round(3*x/w,0)' cut 0,255
```



[0]: 'image.jpg' (640x427x1x3)

## bump2normal

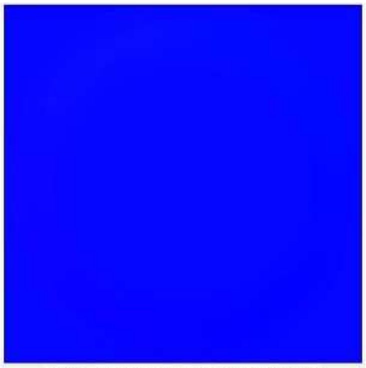
**No arguments**

### Description:

Convert selected bumpmaps to normalmaps.

### Example of use:

```
300,300 circle 50%,50%,128,1,1 blur 5% bump2normal
```



[0]: '[unnamed]' (300x300x1x3)

## camera

**Built-in command**

### Arguments:

- `_camera_index>=0,_nb_frames>0,_skip_frames>=0,_capture_width>=0,_capture_heig`

### Description:

Insert one or several frames from specified camera.

When `nb_frames==0`, the camera stream is released instead of capturing new images.
This command requires features from the OpenCV library (not enabled in G'MIC by default).

## Default values:

`camera_index=0` (default camera), `nb_frames=1`, `skip_frames=0` and `capture_width=capture_height=0` (default size).

---

# canny

## Arguments:

- `_sigma[%]>=0,_low_threshold>=0,_high_threshold>=0`

## Description:

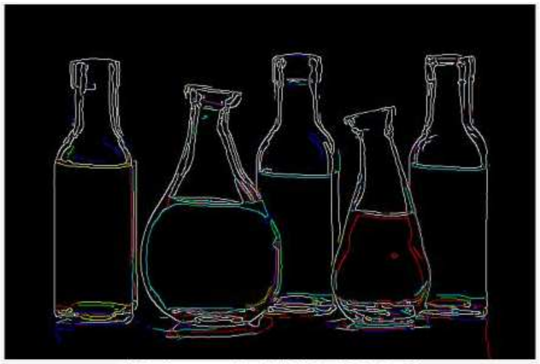
Locate image edges using Canny edge detector.

## Default values:

`sigma=1`, `low_threshold=0.05`, `high_threshold=0.15`.

## Example of use:

```
image.jpg canny 1
```



[0]: 'image.jpg' (640x427x1x3)

---

# cartoon

## Arguments:

- `_smoothness,_sharpening,_threshold>=0,_thickness>=0,_color>=0,quantization>0`

### Description:

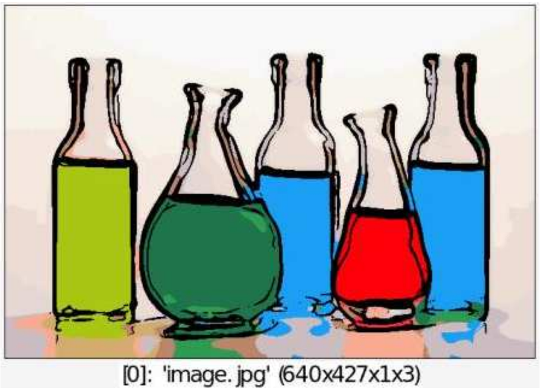
Apply cartoon effect on selected images.

### Default values:

`smoothness=3`, `sharpening=150`, `threshold=20`, `thickness=0.25`, `color=1.5` and `quantization=8`.

### Example of use:

```
image.jpg cartoon 3,50,10,0.25,3,16
```



[0]: 'image.jpg' (640x427x1x3)

## cast

### Arguments:

- `datatype_source,datatype_target`

### Description:

Cast datatype of image buffer from specified source type to specified target type.

`datatype_source` and `datatype_target` can be ***{ uint8 | int8 | uint16 | int16 | uint32 | int32 | uint64 | int64 | float32 | float64 }***.

## cat

### Arguments:

- `filename,_display_line_numbers={ 0 | 1 },_line_selection,`

### Description:

Print specified line selection of given filename on stdout.

### Default values:

`display_line_numbers=1` and `line_selection=^`.

## center3d

**No arguments**

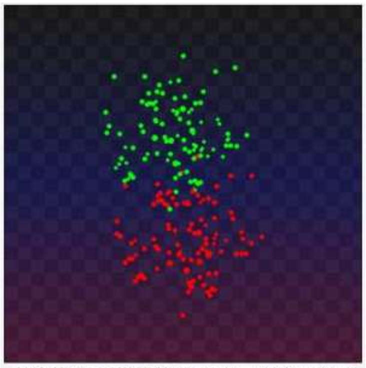
### Description:

Center selected 3D objects at (0,0,0).

(*equivalent to shortcut command* `c3d`).

### Example of use:

```
repeat 100 { circle3d {u(100)},{u(100)},{u(100)},2 } add3d
color3d[-1] 255,0,0 +center3d color3d[-1] 0,255,0 add3d
```



[0]: '[3D circle]' (400 vert., 200 prim.)

## chainring3d

### Arguments:

- `_nb_links>=3,_x_scale>0,_y_scale>0,_z_scale>0`

### Description:

Input 3D chain ring with specified geometry.

`nb_links` should be preferably even.

### Default values:

`nb_links=16`, `x_scale=0.5`, `y_scale=1` and `z_scale=1`.

## Example of use:

```
chainring3d
```

[0]: '[3D chainring]'
(4608 vert., 4608 prim.)

# channels

## Arguments:

- `c0[%],_c1[%]`

## Description:

Keep only specified channels of selected images.

Dirichlet boundary is used when specified channels are out of range.

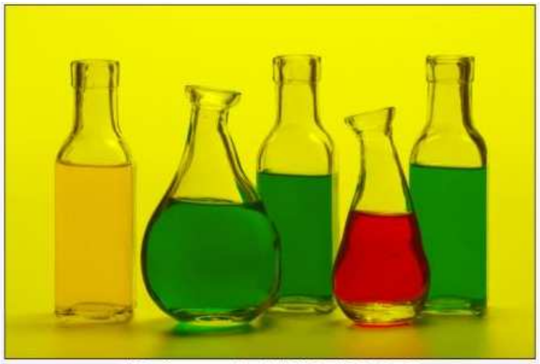
## Default values:

`c1=c0`.

## Examples of use:

### • Example #1

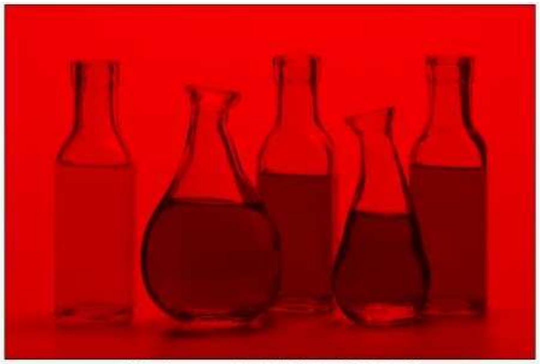
```
image.jpg channels 0,1
```



[0]: 'image.jpg' (640x427x1x2)

- **Example #2**

```
image.jpg luminance channels 0,2
```



[0]: 'image.jpg' (640x427x1x3)

## check

Built-in command

### Arguments:

- `condition`

### Description:

Evaluate specified condition and display an error message if evaluated to false.

## check3d

Built-in command

### Arguments:

- `_is_full_check={ 0 | 1 }`

### Description:

Check validity of selected 3D vector objects, and display an error message

if one of the selected images is not a valid 3D vector object.
Full 3D object check is slower but more precise.

### Default values:

`is_full_check=1`.

## chessboard

### Arguments:

- `size1>0,_size2>0,_offset1,_offset2,_angle,_opacity,_color1,...,_color2,...`

### Description:

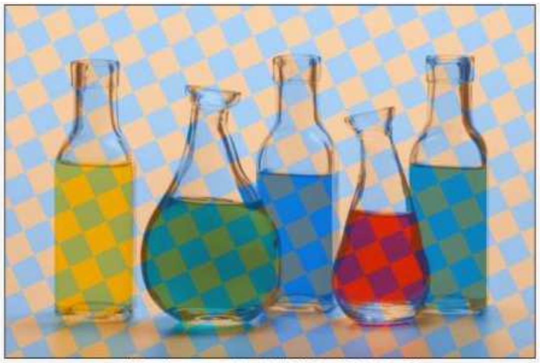
Draw chessboard on selected images.

### Default values:

`size2=size1`, `offset1=offset2=0`, `angle=0`, `opacity=1`, `color1=0` and `color2=255`.

### Example of use:

```
image.jpg chessboard 32,32,0,0,25,0.3,255,128,0,0,128,255
```



[0]: 'image.jpg' (640x427x1x3)

---

## cie1931

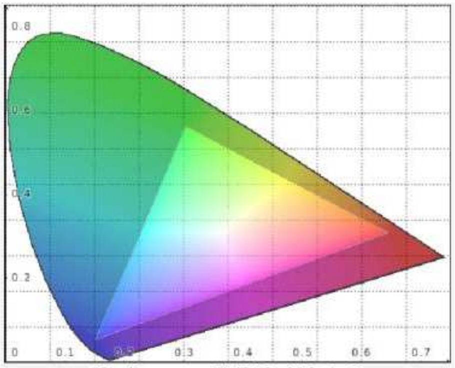
#### No arguments

### Description:

Draw CIE-1931 chromaticity diagram on selected images.

### Example of use:

```
500,400,1,3 cie1931
```



[0]: '[unnamed]' (500x400x1x3)

# circle

## Arguments:

- `x[%],y[%],R[%],_opacity,_pattern,_color1,...`

## Description:

Draw specified colored circle on selected images.

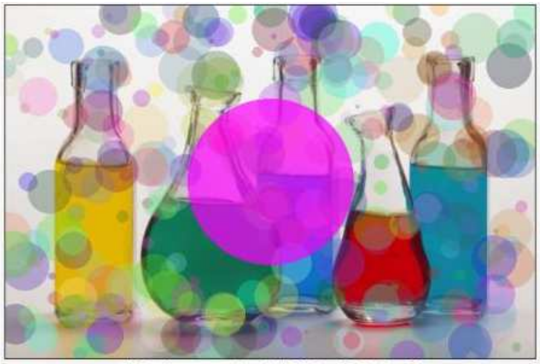
A radius of `100%` stands for `sqrt(width^2+height^2)`.
`pattern` is an hexadecimal number starting with `0x` which can be omitted even if a color is specified. If a pattern is specified, the circle is drawn outlined instead of filled.

## Default values:

`opacity=1`, `pattern=(undefined)` and `color1=0`.

## Example of use:

```
image.jpg repeat 300 circle {u(100)}%,{u(100)}%,{u(30)},0.3,${-rgb}
done circle 50%,50%,100,0.7,255
```



[0]: 'image.jpg' (640x427x1x3)

# circle3d

## Arguments:

- `_x0,_y0,_z0,_radius>=0`

## Description:

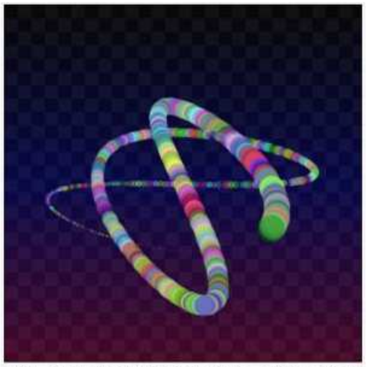
Input 3D circle at specified coordinates.

## Default values:

`x0=y0=z0=0` and `radius=1`.

## Example of use:

```
repeat 500 { a:=$>*pi/250 circle3d {cos(3*$a)},{sin(2*$a)},0,{$a/50}
color3d[-1] ${-rgb},0.4 } add3d
```



[0]: '[3D circle]' (1000 vert., 500 prim.)

---

# circles3d

## Arguments:

- `_radius>=0,_is_outlined={ 0 | 1 }`

## Description:

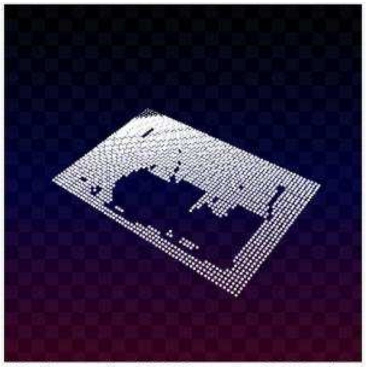
Convert specified 3D objects to sets of 3D circles with specified radius.

## Default values:

`radius=1` and `is_outlined=1`.

## Example of use:

```
image.jpg luminance rescale2d ,40 threshold 50% * 255 pointcloud3d
color3d[-1] 255,255,255 circles3d 0.7
```



[0]: 'image.jpg' (3530 vert., 1765 prim.)

# close_binary

## Arguments:

- `0<=_endpoint_rate<=100,_endpoint_connectivity>=0,_spline_distmax>=0,_segment_`
  `0 | 1 }`

## Description:

Automatically close open shapes in binary images (defining white strokes on black background).

## Default values:

`endpoint_rate=75`, `endpoint_connectivity=2`, `spline_distmax=80`, `segment_distmax=20`, `spline_anglemax=90`, `spline_roundness=1`, `area_min=100`, `allow_self_intersection=1`.

# closing

## Arguments:

- `size>=0` or
- `size_x>=0,size_y>=0,_size_z>=0` or
- `[kernel],_boundary_conditions,_is_real={ 0:binary-mode | 1:real-mode }`

## Description:

Apply morphological closing to selected images.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

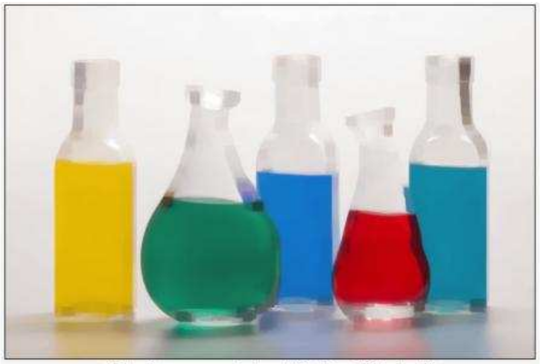
## Default values:

`size_z=1`, `boundary_conditions=1` and `is_real=0`.

## Example of use:

```
image.jpg +closing 10
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# closing_circ

## Arguments:

- `_size>=0,_is_real={ 0 | 1 }`

## Description:

Apply circular dilation of selected images by specified size.

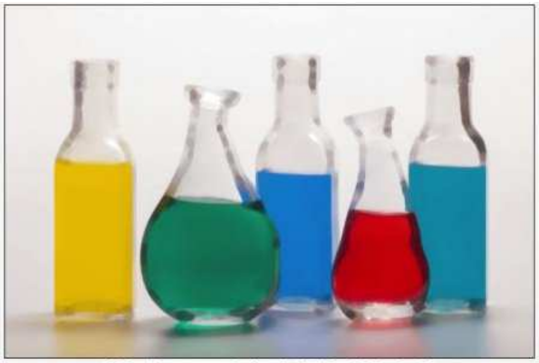
## Default values:

`boundary_conditions=1` and `is_real=0`.

## Example of use:

```
image.jpg +closing_circ 7
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# clut

## Arguments:

- `"clut_name",_resolution>0,_cut_and_round={ 0:no | 1:yes }`

## Description:

Insert one of the 1149 pre-defined CLUTs at the end of the image list.

`clut_name` can be { 12_years_a_slave | 1917 | 2-strip-process | 60s | 60s_faded | 60s_faded_alt | 7drk_21 | action_magenta_01 | action_red_01 | ad_astra | adventure_1453 | agfa_apx_100 | agfa_apx_25 | agfa_precisa_100 | agfa_ultra_color_100 | agfa_vista_200 | agressive_highligjtes_recovery_5 | aladdin | alberto_street | alien_green | ampio | amstragram | amstragram+ | analog_film_1 | analogfx_anno_1870_color | analogfx_old_style_i | analogfx_old_style_ii | analogfx_old_style_iii | analogfx_sepia_color | analogfx_soft_sepia_i | analogfx_soft_sepia_ii | anime | ant-man | apocalypse_this_very_moment | aqua | aqua_and_orange_dark | aquaman | arabica_12 | asistas | atomic_pink | atusa | autumn | autumn_leaves | ava_614 | avalanche | avengers_endgame | azrael_93 | baby_driver | bad_boys_for_life | basuco | bboyz_2 | bc_darkum | beach_aqua_orange | beach_faded_analog | beati | beauty_and_the_beast | berlin_sky | bisogno | black_and_white | black_panther | black_star | black_white_01 | black_white_02 | black_white_03 | black_white_04 | black_white_05 | black_white_06 | blade_runner | bleach_bypass | bleachbypass_1 | bleachbypass_2 | bleachbypass_3 | bleachbypass_4 | bleech_bypass_green | bleech_bypass_yellow_01 | blue_cold_fade | blue_dark | blue_house | blue_ice | blue_love_39 | blue_mono | blue_shadows_01 | bluearchitecture | bluehour | blues | bob_ford | bohemian_rhapsody | bombshell | bourbon_64 | boyado | bright_green_01 | bright_teal_orange | bright_warm | brightgreen | brown_mobster | brownbm | brownish | bw_1 | bw_10 | bw_2 | bw_3 | bw_4 | bw_5 | bw_6 | bw_7 | bw_8 | bw_9 | bw_but_yellow | byers_11 | calidum | candlelight | captain_marvel | caribe | chemical_168 | chrome_01 | cineblue | cinebm_4k | cinema | cinema_2 | cinema_3 | cinema_4 | cinema_5 | cinema_noir | cinematic-1 | cinematic-10 | cinematic-2 | cinematic-3 | cinematic-4 | cinematic-5 | cinematic-6 | cinematic-7 | cinematic-8 | cinematic-9 | cinematic_01 | cinematic_02 | cinematic_03 | cinematic_04 | cinematic_05 | cinematic_06 | cinematic_07 | cinematic_for_flog | cinematic_forest | cinematic_lady_bird | cinematic_mexico | city | city_7 | city_dust | city_of_god | classic_films_01 | classic_films_02 | classic_films_03 | classic_films_04 | classic_films_05 | classic_teal_and_orange | clayton_33 | clear | clear_teal_fade | clouseau_54 | cobi_3 | coffee_44 | cold_clear_blue | cold_clear_blue_1 | cold_ice | cold_simplicity_2 | coldchrome | color_rich | colore | colorful_0209 | colornegative | conflict_01 | contrail_35 | contrast_with_highlights_protection | contrasty_afternoon | contrasty_green | convold | cosa | creed_2 | crispautumn | crispromance | crispwarm | crispwinter | cross_process_cp_130 | cross_process_cp_14 | cross_process_cp_15 | cross_process_cp_16 | cross_process_cp_18 | cross_process_cp_3 | cross_process_cp_4 | cross_process_cp_6 | crushin | cubicle_99 | culor | d_o_1 | dark_blues_in_sunlight | dark_green_02 | dark_green_1 | dark_man_x | dark_orange_teal | dark_place_01 | darkandsomber | darkness | date_39 | day_4nite | day_for_night | day_to_night_kings_blue |

```
deep | deep_blue | deep_dark_warm | deep_high_contrast | deep_teal_fade |
deep_warm_fade | deepskintones_2 | deepskintones_3 | delicatessen |
denoiser_simple_40 | desert_gold_37 | dimension | dimmer | directions_23 |
django_25 | doctor_strange | domingo_145 | dream_1 | dream_85 |
drop_green_tint_14 | dropblues | dunkirk | duotone_blue_red |
earth_tone_boost | eda_0_2 | edgyember | elegance_38 | enchanted | ensaya |
eterna_for_flog | expired_69 | expired_fade | expired_polaroid | extreme |
fade | fade_to_green | faded | faded_47 | faded_alt | faded_analog |
faded_extreme | faded_green | faded_pink-ish | faded_print | faded_retro_01 |
faded_retro_02 | faded_vivid | fadedlook | fallcolors | falua | farkling |
fatos | faux_infrared | faux_infrared_bw_1 | faux_infrared_color_p_2 |
faux_infrared_color_p_3 | faux_infrared_color_r_0a | faux_infrared_color_r_0b
| faux_infrared_color_yp_1 | fezzle | fg_cinebasic | fg_cinebright |
fg_cinecold | fg_cinedrama | fg_cinetealorange_1 | fg_cinetealorange_2 |
fg_cinevibrant | fg_cinewarm | fgcinebasic | fgcinebright | fgcinecold |
fgcinedrama | fgcinetealorange_1 | fgcinetealorange_2 | fgcinevibrant |
fgcinewarm | fight_club | film_0987 | film_9879 | film_gb-19 |
film_high_contrast | film_print_01 | film_print_02 | filmic | filo | flat_30
| flat_blue_moon | flavin | flog_to_rec_709 | foggynight | folger_50 |
ford_v_ferrari | foresta | formula_b | french_comedy | frosted |
frostedbeachpicnic | fuji_160c | fuji_160c_+ | fuji_160c_++ | fuji_160c_- |
fuji_3510_constlclip | fuji_3510_constlmap | fuji_3510_cuspclip |
fuji_3513_constlclip | fuji_3513_constlmap | fuji_3513_cuspclip | fuji_400h |
fuji_400h_+ | fuji_400h_++ | fuji_400h_- | fuji_800z | fuji_800z_+ |
fuji_800z_++ | fuji_800z_- | fuji_astia_100_generic | fuji_astia_100f |
fuji_fp-100c | fuji_fp-100c_+ | fuji_fp-100c_++ | fuji_fp-100c_+++ |
fuji_fp-100c_++_alt | fuji_fp-100c_- | fuji_fp-100c_-- | fuji_fp-100c_alt |
fuji_fp-100c_cool | fuji_fp-100c_cool_+ | fuji_fp-100c_cool_++ |
fuji_fp-100c_cool_- | fuji_fp-100c_cool_-- | fuji_fp-100c_negative |
fuji_fp-100c_negative_+ | fuji_fp-100c_negative_++ | fuji_fp-100c_negative_++
+ | fuji_fp-100c_negative_++_alt | fuji_fp-100c_negative_- |
fuji_fp-100c_negative_-- | fuji_fp-3000b | fuji_fp-3000b_+ | fuji_fp-3000b_++
| fuji_fp-3000b_+++ | fuji_fp-3000b_- | fuji_fp-3000b_-- | fuji_fp-3000b_hc |
fuji_fp-3000b_negative | fuji_fp-3000b_negative_+ | fuji_fp-3000b_negative_++
| fuji_fp-3000b_negative_+++ | fuji_fp-3000b_negative_- |
fuji_fp-3000b_negative_-- | fuji_fp-3000b_negative_early | fuji_fp_100c |
fuji_hdr | fuji_neopan_1600 | fuji_neopan_1600_+ | fuji_neopan_1600_++ |
fuji_neopan_1600_- | fuji_neopan_acros_100 | fuji_provia_100_generic |
fuji_provia_100f | fuji_provia_400f | fuji_provia_400x | fuji_sensia_100 |
fuji_superia_100 | fuji_superia_100_+ | fuji_superia_100_++ |
fuji_superia_100_- | fuji_superia_1600 | fuji_superia_1600_+ |
fuji_superia_1600_++ | fuji_superia_1600_- | fuji_superia_200 |
fuji_superia_200_xpro | fuji_superia_400 | fuji_superia_400_+ |
fuji_superia_400_++ | fuji_superia_400_- | fuji_superia_800 |
fuji_superia_800_+ | fuji_superia_800_++ | fuji_superia_800_- |
fuji_superia_hg_1600 | fuji_superia_reala_100 | fuji_superia_x-tra_800 |
fuji_velvia_100_generic | fuji_velvia_50 | fuji_xtrans_iii_acros |
fuji_xtrans_iii_acros+g | fuji_xtrans_iii_acros+r | fuji_xtrans_iii_acros+ye
| fuji_xtrans_iii_astia | fuji_xtrans_iii_classic_chrome |
fuji_xtrans_iii_mono | fuji_xtrans_iii_mono+g | fuji_xtrans_iii_mono+r |
fuji_xtrans_iii_mono+ye | fuji_xtrans_iii_pro_neg_hi |
fuji_xtrans_iii_pro_neg_std | fuji_xtrans_iii_provia | fuji_xtrans_iii_sepia
```

| fuji_xtrans_iii_velvia | fusion_88 | futuristicbleak_1 | futuristicbleak_2 | futuristicbleak_3 | futuristicbleak_4 | going_for_a_walk | golden | golden_bright | golden_fade | golden_mono | golden_night_softner_43 | golden_sony_37 | golden_vibrant | goldengate | goldentime | goldfx_bright_spring_breeze | goldfx_bright_summer_heat | goldfx_hot_summer_heat | goldfx_perfect_sunset_01min | goldfx_perfect_sunset_05min | goldfx_perfect_sunset_10min | goldfx_spring_breeze | goldfx_summer_heat | good_morning | green_15 | green_2025 | green_action | green_afternoon | green_and_orange | green_blues | green_book | green_conflict | green_day_01 | green_day_02 | green_g_09 | green_indoor | green_light | green_mono | green_yellow | greenish_contrasty | greenish_fade | greenish_fade_1 | gremerta | greyhound | hackmanite | hallowen_dark | happyness_133 | hard_teal_orange | hardboost | harsh_day | harsh_sunset | helios | herderite | heulandite | hiddenite | highlights_protection | hilutite | hitman | hlg_1_1 | honey_light | hong_kong | horrorblue | howlite | huesio | husmes | huyan | hydracore | hyla_68 | hypersthene | hypnosis | hypressen | i_tonya | ideo | ilford_delta_100 | ilford_delta_3200 | ilford_delta_3200_+ | ilford_delta_3200_++ | ilford_delta_3200_- | ilford_delta_400 | ilford_fp_4_plus_125 | ilford_hp_5 | ilford_hp_5_+ | ilford_hp_5_++ | ilford_hp_5_- | ilford_hp_5_plus_400 | ilford_hps_800 | ilford_pan_f_plus_50 | ilford_xp_2 | inception | indoor_blue | industrial_33 | infrared_-_dust_pink | instantc | j | jarklin | jojo_rabbit | joker | jumanji_the_next_level | jurassic_world_fallen_kingdom | justice_league | justpeachy | jwick_21 | k_tone_vintage_kodachrome | kahve_3 | kh_1 | kh_10 | kh_2 | kh_3 | kh_4 | kh_5 | kh_6 | kh_7 | kh_8 | kh_9 | killstreak | kingsman_the_golden_circle | knives_out | kodak_2383_constlclip | kodak_2383_constlmap | kodak_2383_cuspclip | kodak_2393_constlclip | kodak_2393_constlmap | kodak_2393_cuspclip | kodak_bw_400_cn | kodak_e-100_gx_ektachrome_100 | kodak_ektachrome_100_vs | kodak_ektachrome_100_vs_generic | kodak_ektar_100 | kodak_elite_100_xpro | kodak_elite_chrome_200 | kodak_elite_chrome_400 | kodak_elite_color_200 | kodak_elite_color_400 | kodak_elite_extracolor_100 | kodak_hie_hs_infra | kodak_kodachrome_200 | kodak_kodachrome_25 | kodak_kodachrome_64 | kodak_kodachrome_64_generic | kodak_portra_160 | kodak_portra_160_+ | kodak_portra_160_++ | kodak_portra_160_- | kodak_portra_160_nc | kodak_portra_160_nc_+ | kodak_portra_160_nc_++ | kodak_portra_160_nc_- | kodak_portra_160_vc | kodak_portra_160_vc_+ | kodak_portra_160_vc_++ | kodak_portra_160_vc_- | kodak_portra_400 | kodak_portra_400_+ | kodak_portra_400_++ | kodak_portra_400_- | kodak_portra_400_nc | kodak_portra_400_nc_+ | kodak_portra_400_nc_++ | kodak_portra_400_nc_- | kodak_portra_400_uc | kodak_portra_400_uc_+ | kodak_portra_400_uc_++ | kodak_portra_400_uc_- | kodak_portra_400_vc | kodak_portra_400_vc_+ | kodak_portra_400_vc_++ | kodak_portra_400_vc_- | kodak_portra_800 | kodak_portra_800_+ | kodak_portra_800_++ | kodak_portra_800_- | kodak_portra_800_hc | kodak_t-max_100 | kodak_t-max_3200 | kodak_t-max_400 | kodak_tmax_3200 | kodak_tmax_3200_+ | kodak_tmax_3200_++ | kodak_tmax_3200_- | kodak_tmax_3200_alt | kodak_tri-x_400 | kodak_tri-x_400_+ | kodak_tri-x_400_++ | kodak_tri-x_400_- | kodak_tri-x_400_alt | korben_214 | la_la_land | landscape | landscape_01 | landscape_02 | landscape_03 | landscape_04 | landscape_05 | landscape_1 | landscape_10 | landscape_2 | landscape_3 | landscape_4 | landscape_5 | landscape_6 | landscape_7 | landscape_8 | landscape_9 | lateafternoonwanderlust | latesunset | lavark | lc_1 | lc_10 |

```
lc_2 | lc_3 | lc_4 | lc_5 | lc_6 | lc_7 | lc_8 | lc_9 | lenox_340 | levex |
life_giving_tree | light | light_blown | litore | little_women | logan | lomo
| lomography_redscale_100 | lomography_x-pro_slide_200 | london_nights |
longbeachmorning | loro | lotta | louetta | low_contrast_blue | low_key_01 |
lucky_64 | lushgreen | lushgreensummer | mad_max_fury_road | maesky |
magenta_day | magenta_day_01 | magenta_dream | magenta_yellow | magentacoffee
| magichour | marriage_story | matrix | mckinnon_75 | memories | mercato |
metropolis | milo_5 | minimalistcaffeination | modern_film | modern_films_01
| modern_films_02 | modern_films_03 | modern_films_04 | modern_films_05 |
modern_films_06 | modern_films_07 | molti | mono_2 | mono_tinted | monochrome
| monochrome_1 | monochrome_2 | moody_1 | moody_10 | moody_2 | moody_3 |
moody_4 | moody_5 | moody_6 | moody_7 | moody_8 | moody_9 | moonlight |
moonlight_01 | moonlight_2 | moonrise | morning_6 | morroco_16 | mostly_blue
| mother! | motus | moviz_1 | moviz_10 | moviz_11 | moviz_12 | moviz_13 |
moviz_14 | moviz_15 | moviz_16 | moviz_17 | moviz_18 | moviz_19 | moviz_2 |
moviz_20 | moviz_21 | moviz_22 | moviz_23 | moviz_24 | moviz_25 | moviz_26 |
moviz_27 | moviz_28 | moviz_29 | moviz_3 | moviz_30 | moviz_31 | moviz_32 |
moviz_33 | moviz_34 | moviz_35 | moviz_36 | moviz_37 | moviz_38 | moviz_39 |
moviz_4 | moviz_40 | moviz_41 | moviz_42 | moviz_43 | moviz_44 | moviz_45 |
moviz_46 | moviz_47 | moviz_48 | moviz_5 | moviz_6 | moviz_7 | moviz_8 |
moviz_9 | mucca | mute_shift | muted_01 | muted_fade | mysticpurplesunset |
nah | natural_vivid | naturalboost | negative | nemesis | neon_770 | neutral
| neutral_pump | neutral_teal_orange | neutral_warm_fade | newspaper |
night_01 | night_02 | night_03 | night_04 | night_05 | night_blade_4 |
night_king_141 | night_spy | night_view | nightfromday | nightlife | nigrum |
no_time_to_die | nostalgiahoney | nostalgic | nw-1 | nw-10 | nw-2 | nw-3 |
nw-4 | nw-5 | nw-6 | nw-7 | nw-8 | nw-9 | old_west | once_upon_a_time |
once_upon_a_time_in_hollywood | onda | only_red | only_red_and_blue |
operation_yellow | orange_dark_4 | orange_dark_7 | orange_dark_look |
orange_tone | orange_underexposed | orangeandblue | oranges | padre | paladin
| paladin_1875 | parasite | partia | pasadena_21 | passing_by | perso |
picola | pink_fade | pirates_of_the_caribbean | pitaya_15 | pmcinematic_01 |
pmcinematic_02 | pmcinematic_03 | pmcinematic_04 | pmcinematic_05 |
pmcinematic_06 | pmcinematic_07 | pmnight_01 | pmnight_02 | pmnight_03 |
pmnight_04 | pmnight_05 | polaroid_664 | polaroid_665 | polaroid_665_+ |
polaroid_665_++ | polaroid_665_- | polaroid_665_-- | polaroid_665_negative |
polaroid_665_negative_+ | polaroid_665_negative_- | polaroid_665_negative_hc
| polaroid_667 | polaroid_669 | polaroid_669_+ | polaroid_669_++ |
polaroid_669_+++ | polaroid_669_- | polaroid_669_-- | polaroid_669_cold |
polaroid_669_cold_+ | polaroid_669_cold_- | polaroid_669_cold_-- |
polaroid_672 | polaroid_690 | polaroid_690_+ | polaroid_690_++ |
polaroid_690_- | polaroid_690_-- | polaroid_690_cold | polaroid_690_cold_+ |
polaroid_690_cold_++ | polaroid_690_cold_- | polaroid_690_cold_-- |
polaroid_690_warm | polaroid_690_warm_+ | polaroid_690_warm_++ |
polaroid_690_warm_- | polaroid_690_warm_-- | polaroid_polachrome |
polaroid_px-100uv+_cold | polaroid_px-100uv+_cold_+ |
polaroid_px-100uv+_cold_++ | polaroid_px-100uv+_cold_+++ |
polaroid_px-100uv+_cold_- | polaroid_px-100uv+_cold_-- |
polaroid_px-100uv+_warm | polaroid_px-100uv+_warm_+ |
polaroid_px-100uv+_warm_++ | polaroid_px-100uv+_warm_+++ |
polaroid_px-100uv+_warm_- | polaroid_px-100uv+_warm_-- | polaroid_px-680 |
polaroid_px-680_+ | polaroid_px-680_++ | polaroid_px-680_- |
polaroid_px-680_-- | polaroid_px-680_cold | polaroid_px-680_cold_+ |
polaroid_px-680_cold_++ | polaroid_px-680_cold_++_alt |
polaroid_px-680_cold_- | polaroid_px-680_cold_-- | polaroid_px-680_warm |
polaroid_px-680_warm_+ | polaroid_px-680_warm_++ | polaroid_px-680_warm_- |
polaroid_px-680_warm_-- | polaroid_px-70 | polaroid_px-70_+ |
polaroid_px-70_++ | polaroid_px-70_+++ | polaroid_px-70_- | polaroid_px-70_--
| polaroid_px-70_cold | polaroid_px-70_cold_+ | polaroid_px-70_cold_++ |
polaroid_px-70_cold_- | polaroid_px-70_cold_-- | polaroid_px-70_warm |
polaroid_px-70_warm_+ | polaroid_px-70_warm_++ | polaroid_px-70_warm_- |
polaroid_px-70_warm_-- | polaroid_time_zero_expired |
polaroid_time_zero_expired_+ | polaroid_time_zero_expired_++ |
polaroid_time_zero_expired_- | polaroid_time_zero_expired_-- |
polaroid_time_zero_expired_--- | polaroid_time_zero_expired_cold |
polaroid_time_zero_expired_cold_- | polaroid_time_zero_expired_cold_-- |
polaroid_time_zero_expired_cold_--- | portrait | portrait_1 | portrait_10 |
portrait_2 | portrait_3 | portrait_4 | portrait_5 | portrait_6 | portrait_7 |
portrait_8 | portrait_9 | progressen | protect_highlights_01 | prussian_blue
| pseudogrey | purple | purple_2 | quraqqq_12 | randas | red_afternoon_01 |
red_day_01 | red_dream_01 | redblueyellow | reds | reds_oranges_yellows |
reeve_38 | remy_24 | rest_33 | retro | retro_brown_01 | retro_magenta_01 |
retro_summer_3 | retro_yellow_01 | rocketman | rollei_ir_400 |
rollei_ortho_25 | rollei_retro_100_tonal | rollei_retro_80s | rotate_muted |
rotate_vibrant | rotated | rotated_crush | satid | saturated_blue |
saving_private_damon | scala | science_fiction | scrittle | sea | seges |
selor | sensum | separation | serenity | seringe_4 | serpent |
seventies_magazine | sevsuz | shade_kings_ink | shadow_king_39 | shine |
sicario | sino | skin_tones | slog_to_rec709_basic | slog_to_rec709_contrasty
| slog_to_rec709_crush_shadows | slog_to_rec709_green_correction |
smart_contrast | smokey | smooth_clear | smooth_cromeish | smooth_fade |
smooth_green_orange | smooth_sailing | smooth_teal_orange | soft_fade |
softblackandwhite | softwarming | solarized_color | solarized_color_2 | soldi
| spider-man_far_from_home | spotlight | springmorning | sprocket_231 |
spy_29 | standard | star_wars_the_rise_of_skywalker | strano | street |
stringa | studio_skin_tone_shaper | subtle_blue | subtle_green |
subtle_yellow | sully | summer | summer_alt | sunlight_love_11 | sunlightlove
| sunny | sunny_alt | sunny_rich | sunny_warm | sunset | sunset_aqua_orange |
sunset_intense_violet_blue | sunset_violet_mood | super_warm |
super_warm_rich | sutro_fx | sweet_bubblegum | sweet_gelatto | taşdemirrr_1 |
taiga | tarraco | teal-orange_for_flog | teal_fade | teal_moonlight |
tealmagentagold | tealorange | tealorange_1 | tealorange_2 | tealorange_3 |
technicalfx_backlight_filter | teigen_28 | tenet | tensiongreen_1 |
tensiongreen_2 | tensiongreen_3 | tensiongreen_4 | terra_4 | the_dark_knight
| the_darkest_hour | the_gentelmen | the_grand_budapest_hotel |
the_hurt_locker | the_irishman | the_lighthouse | the_lobster | the_martian |
the_matrices | the_revenant | the_shape_of_water | the_social_network |
the_two_popes | the_way_back | thor_ragnarok | thriller_2 | tirare |
toastedgarden | top_gun_maverick | trent_18 | true_colors_8 | turkiest_42 |
tutto | tweed_71 | ultra_water | uncut_gems | undeniable | undeniable_2 |
underwater | unknown | upglow | urban_01 | urban_02 | urban_03 | urban_04 |
urban_05 | urban_cowboy | uzbek_bukhara | uzbek_marriage | uzbek_samarcande |
valize | valsky | velvetia | venom | very_warm_greenish | vfb_21 | vibrant |
vibrant_alien | vibrant_contrast | vibrant_cromeish | victory | vintage |
```

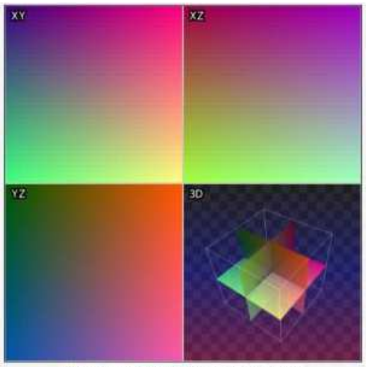
*vintage_01 | vintage_02 | vintage_03 | vintage_04 | vintage_05 | vintage_163 | vintage_alt | vintage_brighter | vintage_chrome | vintage_mob | vintage_warmth_1 | violet_taste | vireo_37 | vita | vivid | vubes | war_for_the_planet_of_the_apes | warm | warm_dark_contrasty | warm_fade | warm_fade_1 | warm_highlight | warm_neutral | warm_sunset_red | warm_teal | warm_vintage | warm_yellow | wavefire | waves | well_see | western | western_6 | westernlut_2 | westernlut_2_13 | whiter_whites | winterlighthouse | wipe | wolf_of_wall_street | wonder_woman | wooden_gold_20 | x-men_dark_phoenix | yangabuz_8 | yellow_55b | yellow_film_01 | yellowstone | you_can_do_it | zed_32 | zeke_39 | zilverfx_bw_solarization | zilverfx_infrared | zilverfx_vintage_bw | zombieland_double_tap }*

## Default values:

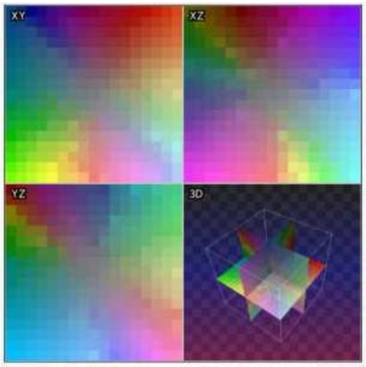
`resolution=33` and `cut_and_round=1`.

## Example of use:

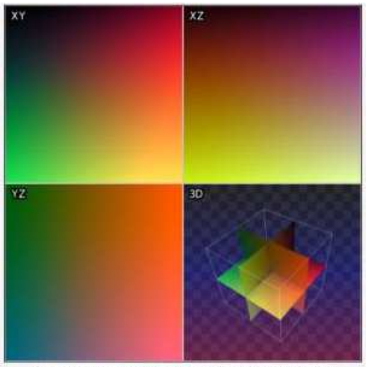
```
clut summer clut alien_green,17 clut orange_dark4,48
```



[0]: 'summer' (33x33x33x3)



[1]: 'alien_green' (17x17x17x3)



[2]: 'orange_dark_4' (48x48x48x3)

# clut2hald

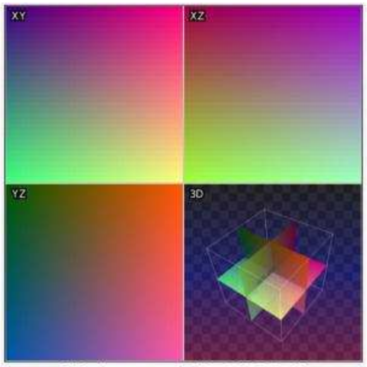
**No arguments**

## Description:

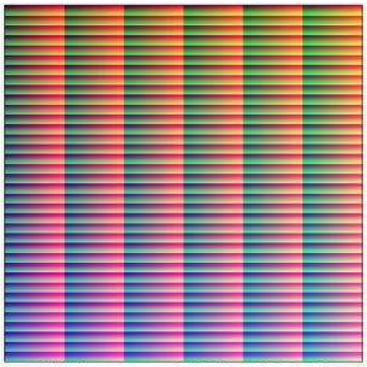
Convert selected 3D CLUTs to 2D HaldCLUTs.

## Example of use:

```
clut summer +clut2hald
```



[0]: 'summer' (33x33x33x3)



[1]: 'summer_c1' (216x216x1x3)

## cmy2rgb

**No arguments**

### Description:

Convert color representation of selected images from CMY to RGB.

## cmyk2rgb

**No arguments**

### Description:

Convert color representation of selected images from CMYK to RGB.

## color2name

### Arguments:

- `R,G,B`

### Description:

Return the name (as a string, in English) that most matches the specified color.

## color3d

### Arguments:

- `R,_G,_B,_opacity`

## Description:

Set color (and optionally opacity) of selected 3D objects.

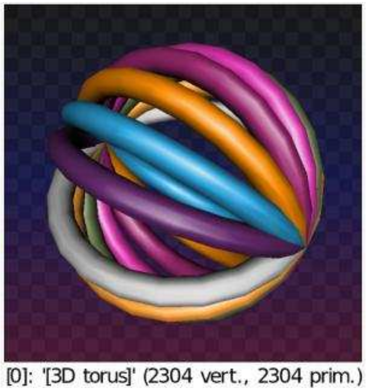
(*equivalent to shortcut command* `col3d`).

## Default values:

`B=G=R` and `opacity=(undefined)`.

## Example of use:

```
torus3d 100,10 double3d 0 repeat 7 { +rotate3d[-1] 1,0,0,20
color3d[-1] ${-rgb} } add3d
```



[0]: '[3D torus]' (2304 vert., 2304 prim.)

# color_ellipses

## Arguments:

- `_count>0,_radius>=0,_opacity>=0`

## Description:

Add random color ellipses to selected images.

## Default values:

`count=400`, `radius=5` and `opacity=0.1`.

## Example of use:

```
image.jpg +color_ellipses ,,0.15
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

# colorblind

## Arguments:

- `type={ 0:protanopia | 1:protanomaly | 2:deuteranopia | 3:deuteranomaly | 4:tritanopia | 5:tritanomaly | 6:achromatopsia | 7:achromatomaly }`

## Description:

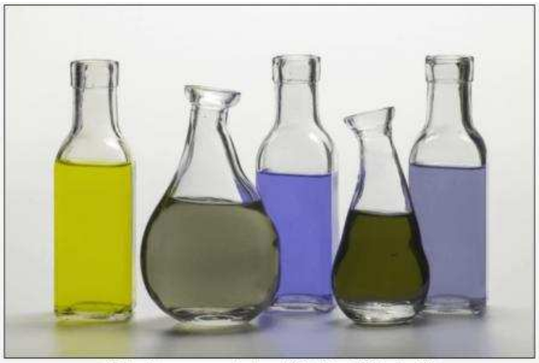
Simulate color blindness vision.

Simulation method of Vienot, Brettel & Mollon 1999, "Digital video colourmaps for checking the legibility of displays by dichromats".
The dichromacy matrices of the paper were adapted to sRGB (RGB->XYZ).
Anomalous trichromacy simulated via linear interpolation with the identity and a factor of 0.6.

## Example of use:

```
image.jpg +colorblind 0
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# colorcube3d

## Arguments:

- `_is_wireframe={ 0 | 1 }`

## Description:

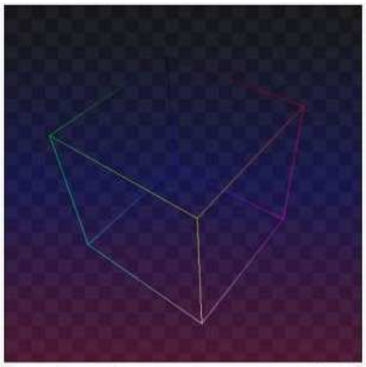
Input 3D color cube.

## Default values:

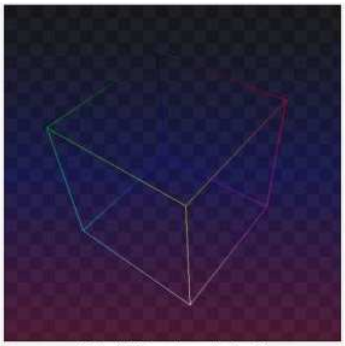
`is_wireframe=0`.

## Example of use:

```
colorcube3d mode3d 2 +primitives3d 1
```



[0]: '[3D colorcube]' (8 vert., 12 prim.)



[1]: '[3D colorcube]_c1'
(8 vert., 12 prim.)

---

# colorize3d

## Arguments:

- `_color_function,_passed_images_for_color_function`

## Description:

Colorize primitives of selected 3D objects, according to a specified function.

- `color_function` returns a G,GA,RGB or RGBA vector that can depend on variables `x`, `y` and `z`, which are defined as the barycenter coordinates for each primitive.
- `passed_images_for_color_function` can be specified as a selection (e.g. `[0,2]`) of images that will be inserted at the end of the image list while modifying 3D objects, so that the `color_function` can have access to their content.

## Default values:

`color_function=[x,y,z]` and `passed_images_for_color_function=`.

## Example of use:

```
torus3d 100,40,640,100 c3d n3d mul3d 256 +3d 128,128,128 sample
colorful,257 colorize3d[0] "I(#-1,x,y,0)",[1]
```

[0]: '[3D torus]'
(64000 vert., 64000 prim.)

[1]: 'colorful' (257x257x1x3)

# colormap

## Arguments:

- `nb_levels>=0,_method={ 0:median-cut | 1:k-means },_sort_vectors`

## Description:

Estimate best-fitting colormap with `nb_colors` entries, to index selected images.

Set `nb_levels==0` to extract all existing colors of an image.
`sort_vectors` can be ***{ 0:unsorted | 1:by increasing norm | 2:by decreasing occurrence }***.

## Default values:

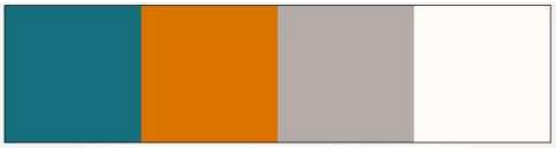
`method=1` and `sort_vectors=1`.

This command has a **tutorial page**.

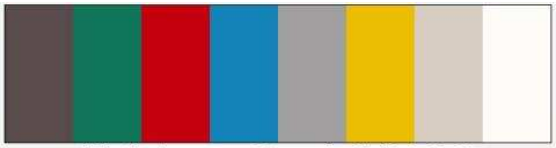
## Example of use:

```
image.jpg +colormap[0] 4 +colormap[0] 8 +colormap[0] 16
```

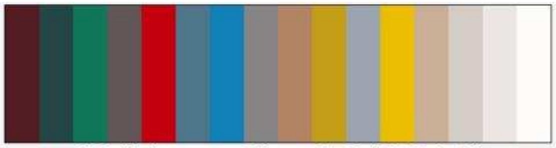
[0]: 'image.jpg' (640x427x1x3)



[1]: '[colormap of image]_c1' (4x1x1x3)



[2]: '[colormap of image]_c1' (8x1x1x3)



[3]: '[colormap of image]_c1' (16x1x1x3)

## columns

### Arguments:

- `x0[%],_x1[%]`

### Description:

Keep only specified columns of selected images.

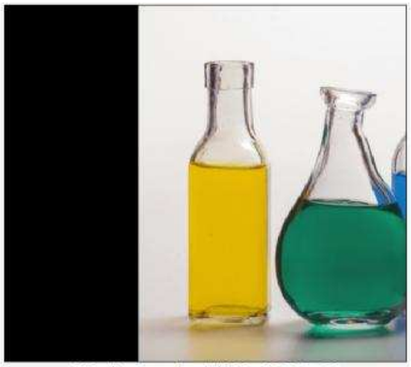
Dirichlet boundary is used when specified columns are out of range.

### Default values:

`x1=x0`.

### Example of use:

```
image.jpg columns -25%,50%
```



[0]: 'image.jpg' (481x427x1x3)

## command

Built-in command

### Arguments:

- `_add_debug_info={ 0 | 1 },{ filename | http[s]://URL | "string" }`

### Description:

Import G'MIC custom commands from specified file, URL or string.

(*equivalent to shortcut command* `m`).

Imported commands are available directly after the `command` invocation.
Specified filename is not allowed to contain colons `:`.

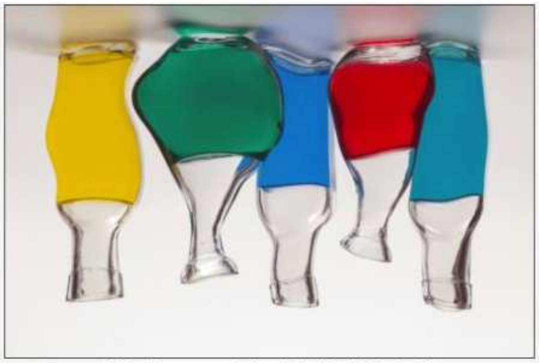
### Default values:

`add_debug_info=1` (except for a "string" argument, in which case `add_debug_info=0`).

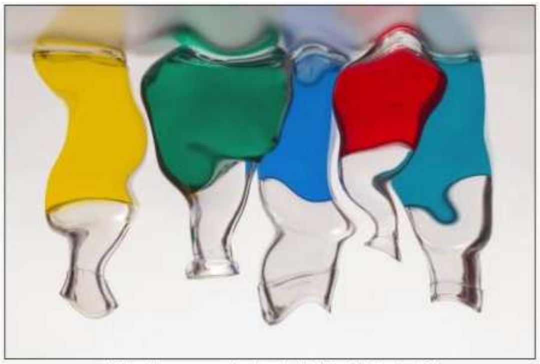
### Example of use:

```
image.jpg command "foo : mirror y deform $""1" +foo[0] 5 +foo[0] 15
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

## complex2polar

**No arguments**

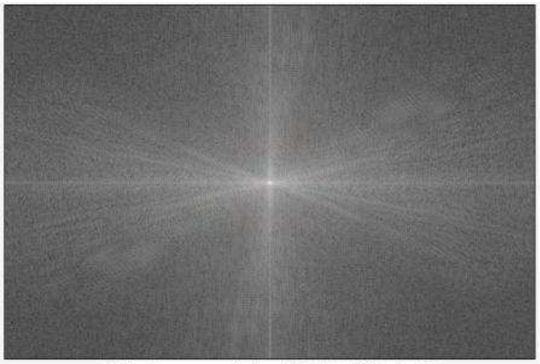
### Description:

Compute complex to polar transforms of selected images.

### Example of use:

```
image.jpg +fft complex2polar[-2,-1] log[-2] shift[-2] 50%,50%,0,0,2
remove[-1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## compose_channels

**No arguments**

### Description:

Compose all channels of each selected image, using specified arithmetic operator (+,-,or,min,...).

### Default values:

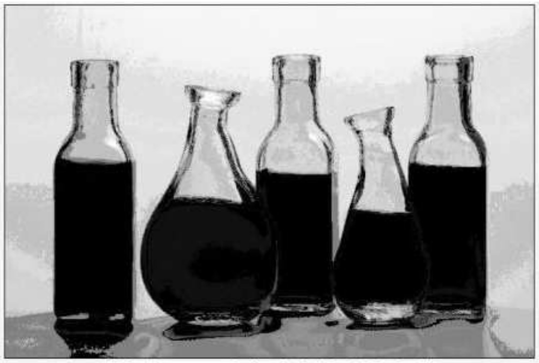
`1=+`.

This command has a **tutorial page**.

### Example of use:

```
image.jpg +compose_channels and
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x1)

## compose_freq

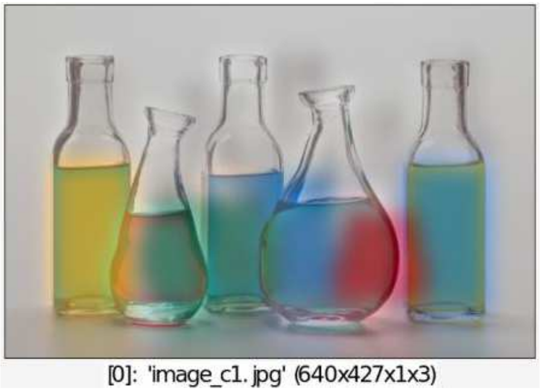
**No arguments**

### Description:

Compose selected low and high frequency parts into new images.

## Example of use:

```
image.jpg split_freq 2% mirror[-1] x compose_freq
```



[0]: 'image_c1.jpg' (640x427x1x3)

# compress_clut

### No arguments

## Description:

Compress selected color LUTs as sequences of colored keypoints.

# compress_huffman

## Arguments:

- `[huffman_tree],_max_leaf_value`

## Description:

Compress selected images with Huffman coding.

## See also:

**decompress_huffman**, **huffman_tree**.

# compress_rle

## Arguments:

- `_is_binary_data={ 0 | 1 },_maximum_sequence_length>=0`

## Description:

Compress selected images as 2xN data matrices, using RLE algorithm.

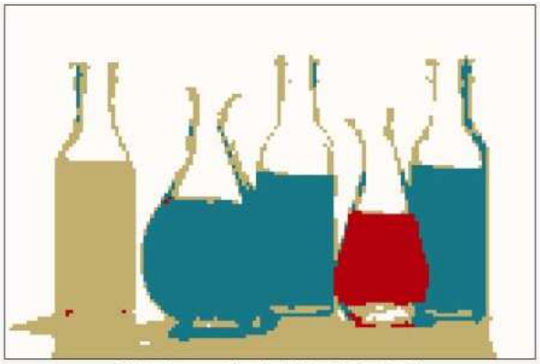
Set `maximum_sequence_length=0` to disable maximum length constraint.

## Default values:

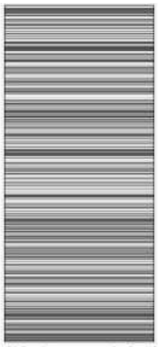
`is_binary_data=0` and `maximum_sequence_length=0`.

## Example of use:

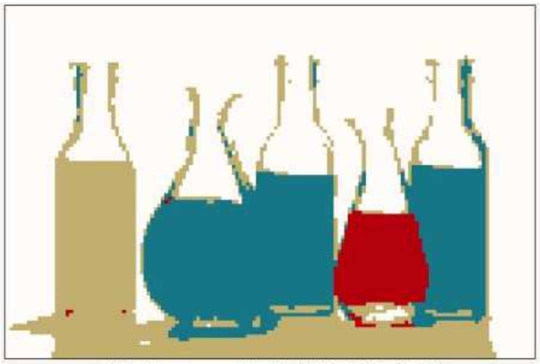
```
image.jpg rescale2d ,100 quantize 4 round +compress_rle ,
+decompress_rle[-1]
```



[0]: 'image.jpg' (150x100x1x3)



[1]: 'image_c1.jpg' (1x6040x1x1)



[2]: '[unnamed]_c1' (150x100x1x3)

# compress_to_keypoints

## Arguments:

- `_method,_max_keypoints>=0,_err_avg[%]>=0,_err_max[%]>=0,_"err_command"`

## Description:

Compress each of the selected images into a set of keypoints that can be further decompressed using command **decompress_from_keypoints**.
**Beware**: This type of compression is effective only for images with very smooth content.
`method` can be ***{ 0:PDE | 1:RBF }***. Add `2` to `method` to enable removal step.

- `max_keypoints` is the maximal number of keypoints generated by the compression method. If `max_keypoints<0`, the removal step is not done when number of maximal keypoints has been reached. `max_keypoints=0` means 'no limits'.
- `err_avg` is the desired average compression error.
- `err_max` is the desired pointwise max compression error.
- `err_command` is the code of a command that inputs the two images `[reference]` and `[compressed]` and compute a single error map as a last image.

Defaults values: `method=3`, `max_keypoints=0`, `err_avg=1%`, `err_max=5%` and 'err_command=-. [0] norm.'

# cone3d

## Arguments:

- `_radius,_height,_nb_subdivisions>0`

## Description:

Input 3D cone at (0,0,0), with specified geometry.

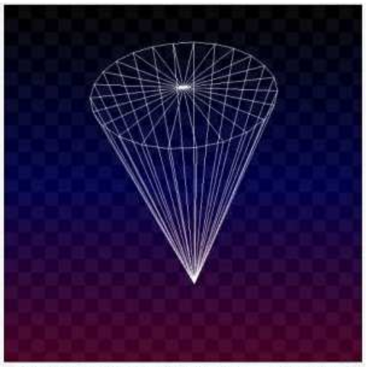
## Default values:

`radius=1`, `height=1` and `nb_subdivisions=24`.

## Example of use:

```
cone3d 10,40 +primitives3d 1 color3d[-2] ${-rgb}
```

[0]: '[3D cone]' (26 vert., 48 prim.)



[1]: '[3D cone]_c1' (26 vert., 72 prim.)

# continue

Built-in command

## No arguments

## Description:

Go to end of current `do...while`, `for...done`, `foreach...done`, `local...done` or `repeat...done` block.

## Example of use:

```
image.jpg repeat 10 blur 1 if 1==1 continue fi deform 10 done
```

[0]: 'image.jpg' (640x427x1x3)

---

# convolve

**Built-in command**

## Arguments:

- `[mask],_boundary_conditions,_is_normalized={ 0 | 1 },_channel_mode,_xcenter,_ycenter,_zcenter,_xstart,_ystart,_zstart,_xend,_yen`

## Description:

Convolve selected images by specified mask.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`channel_mode` can be ***{ 0:all | 1:one-for-one | 2:partial sum | 3:full sum }***.
`interpolation_type` can be ***{ 0:nearest-neighbor | 1:linear }***.

## Default values:

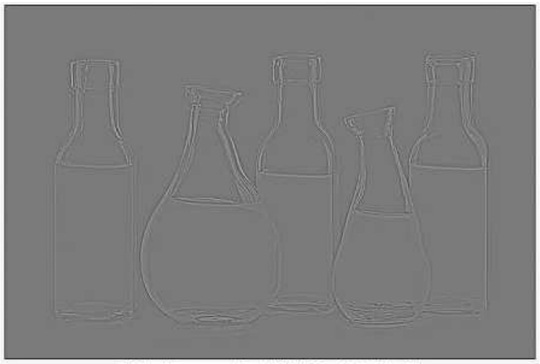
`boundary_conditions=1`, `is_normalized=0`, `channel_mode=1`, `xcenter=ycenter=zcenter=(undefined)`, `xstart=ystart=zstart=0`, `xend=yend=zend=(max-coordinates)`, `xstride=ystride=zstride=1`, `xdilation=ydilation=zdilation=1` and `interpolation_type=0`.

This command has a **tutorial page**.

## Examples of use:

- **Example #1**

```
image.jpg (0,1,0;1,-4,1;0,1,0) convolve[-2] [-1] keep[-2]
```

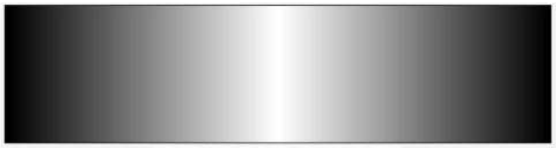


[0]: 'image.jpg' (640x427x1x3)

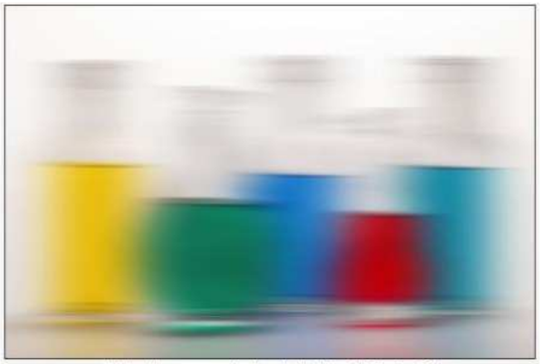
- **Example #2**

```
image.jpg (0,1,0) resize[-1] 130,1,1,1,3 +convolve[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '(0,1,0)' (130x1x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

## convolve_fft

### Arguments:

- `[mask],_boundary_conditions`

### Description:

Convolve selected images with specified mask, in the fourier domain.

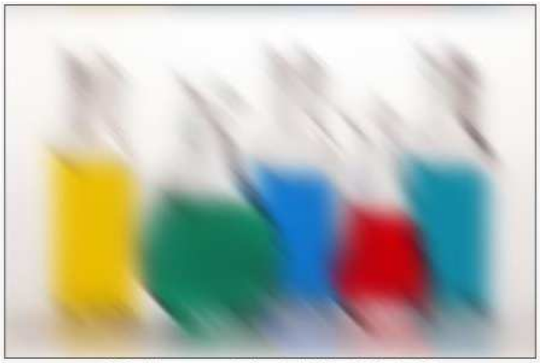
`boundary_conditions` can be *`{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`*.

### Example of use:

```
image.jpg 100%,100% gaussian[-1] 20,1,45 +convolve_fft[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)

[1]: '[unnamed]' (640x427x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

## correlate

Built-in command

### Arguments:

- `[mask],_boundary_conditions,_is_normalized={ 0 | 1 },_channel_mode,_xcenter,_ycenter,_zcenter,_xstart,_ystart,_zstart,_xend,_yen`

### Description:

Correlate selected images by specified mask.

`boundary_conditions` can be *`{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`*.
`channel_mode` can be *`{ 0:all | 1:one-for-one | 2:partial sum | 3:full sum }`*.
`interpolation_type` can be *`{ 0:nearest-neighbor | 1:linear }`*.

### Default values:

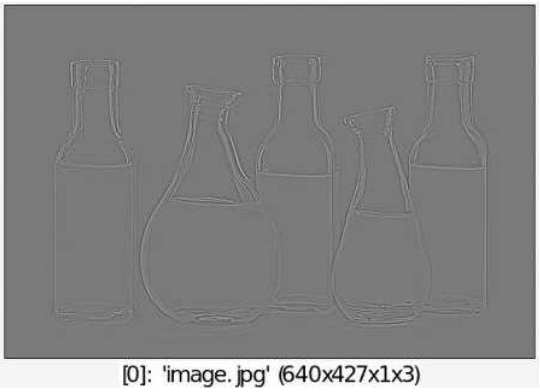
`boundary_conditions=1`, `is_normalized=0`, `channel_mode=1`, `xcenter=ycenter=zcenter=-1`, `xstart=ystart=zstart=0`, `xend=yend=zend=(max-`

`coordinates)`, `xstride=ystride=zstride=1`, `xdilation=ydilation=zdilation=1` and `interpolation_type=0`.

## Examples of use:

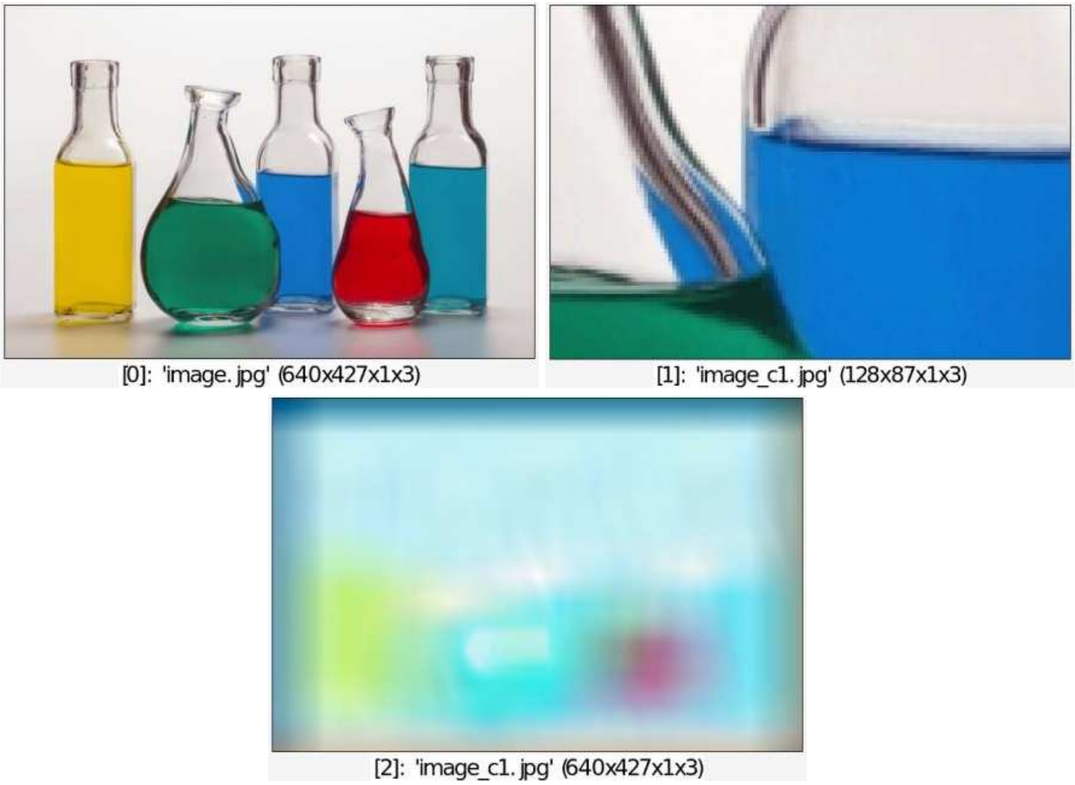
### • Example #1

```
image.jpg (0,1,0;1,-4,1;0,1,0) correlate[-2] [-1] keep[-2]
```


[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +crop 40%,40%,60%,60% +correlate[0] [-1],0,1
```


[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (128x87x1x3)

[2]: 'image_c1.jpg' (640x427x1x3)

## cos

Built-in command

### No arguments

## Description:

Compute the pointwise cosine of selected images.

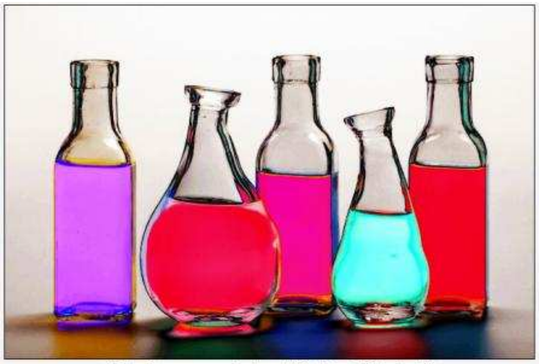
This command has a **tutorial page**.

## Examples of use:

### • Example #1

```
image.jpg +normalize 0,{2*pi} cos[-1]
```

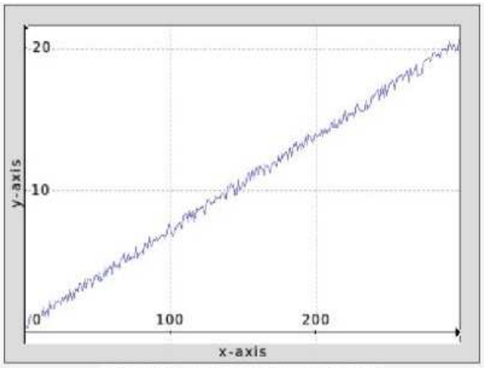
[0]: 'image.jpg' (640x427x1x3)



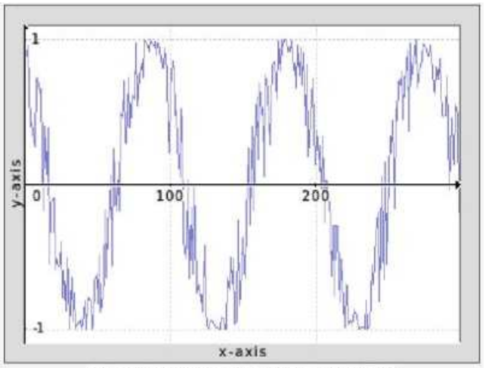
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'20*x/w+u' +cos display_graph 400,300
```



[0]: '[20*x/w+u]' (400x300x1x3)



[1]: '[20*x/w+u]_c1' (400x300x1x3)

# cosh

**Built-in command**

### No arguments

## Description:

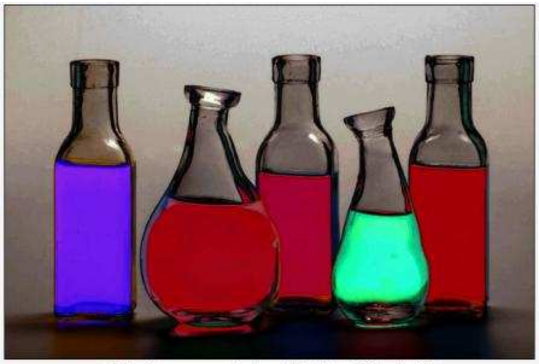
Compute the pointwise hyperbolic cosine of selected images.

## Examples of use:

### • Example #1

```
image.jpg +normalize -3,3 cosh[-1]
```

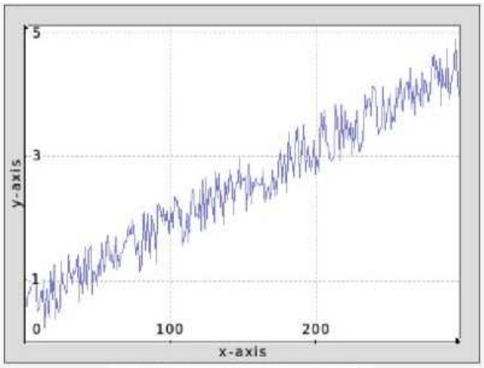
[0]: 'image.jpg' (640x427x1x3)



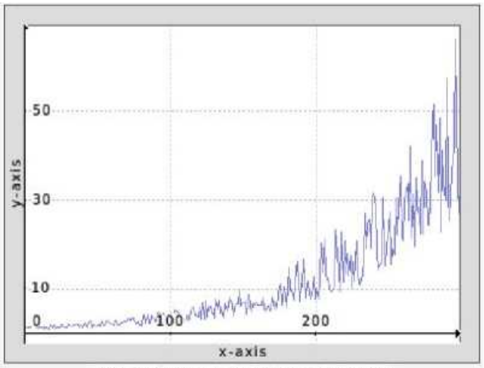
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'4*x/w+u' +cosh display_graph 400,300
```



[0]: '[4*x/w+u]' (400x300x1x3)



[1]: '[4*x/w+u]_c1' (400x300x1x3)

# count_colors

## Arguments:

- `_count_until={ 0` or
- `none | >0` or
- `max number of counted colors }`

## Description:

Count number of distinct colors in selected images until it reaches the specified max number of counted colors.

Set `count_until` to `0` to disable limit on counted colors.
This command returns the number of distinct colors for each image (separated by commas).

# covariance_vectors

## Arguments:

- `_avg_outvarname`

## Description:

Return the covariance matrix of the vector-valued colors in the latest of the selected images

(for arbitrary number of channels).
Parameter `avg_outvarname` is used as a variable name that takes the value of the average vector-value.

# cracks

## Arguments:

- `0<=_density<=100,_is_relief={ 0 | 1 },_opacity,_color1,...`

## Description:

Draw random cracks on selected images with specified color.

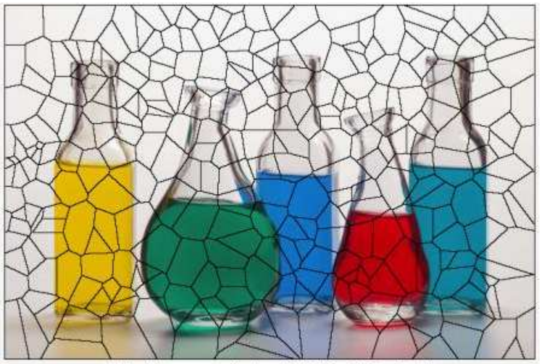
## Default values:

`density=25`, `is_relief=0`, `opacity=1` and `color1=0`.

## Example of use:

```
image.jpg +cracks ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# crop

Built-in command

## Arguments:

- `x0[%],x1[%],_boundary_conditions` or
- `x0[%],y0[%],x1[%],y1[%],_boundary_conditions` or
- `x0[%],y0[%],z0[%],x1[%],y1[%],z1[%],_boundary_conditions` or
- `x0[%],y0[%],z0[%],c0[%],x1[%],y1[%],z1[%],c1[%],_boundary_conditions`

## Description:

Crop selected images with specified region coordinates.

(*equivalent to shortcut command* `z`).

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

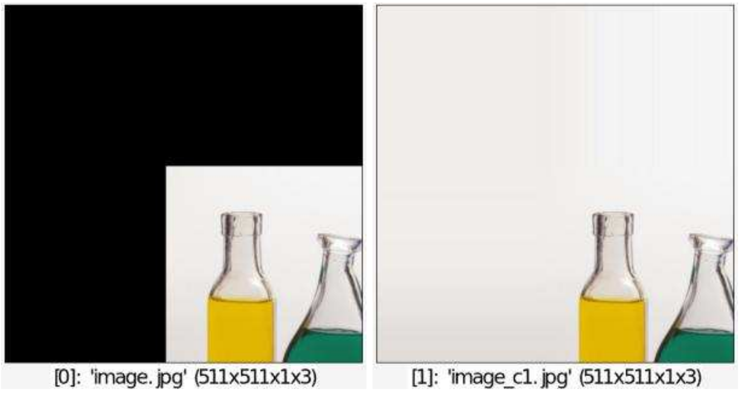
## Default values:

`boundary_conditions=0`.

## Examples of use:

### • Example #1

```
image.jpg +crop -230,-230,280,280,1 crop[0] -230,-230,280,280,0
```


[0]: 'image.jpg' (511x511x1x3)
[1]: 'image_c1.jpg' (511x511x1x3)

### • Example #2

```
image.jpg crop 25%,25%,75%,75%
```

[0]: 'image.jpg' (320x214x1x3)

# cross_correlation

## Arguments:

- `[mask]`

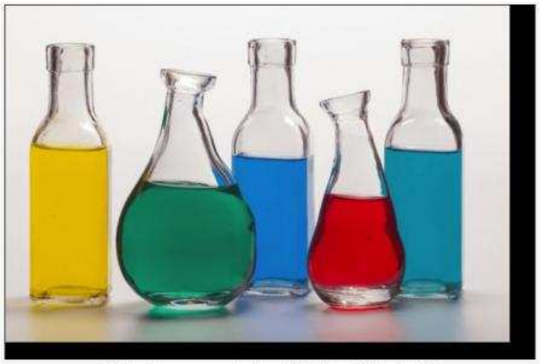
## Description:

Compute cross-correlation of selected images with specified mask.

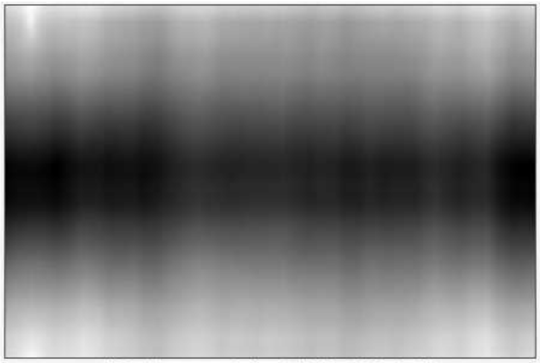
## Example of use:

```
image.jpg +shift -30,-20 +cross_correlation[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x1)

# cubes3d

## Arguments:

- `_size>=0`

## Description:

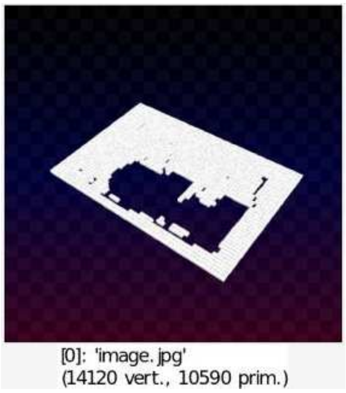
Convert specified 3D objects to sets of 3D cubes with specified size.

## Default values:

`size=1`.

## Example of use:

```
image.jpg luminance rescale2d ,40 threshold 50% * 255 pointcloud3d
color3d[-1] 255,255,255 cubes3d 1
```



[0]: 'image.jpg'
(14120 vert., 10590 prim.)

---

# cubism

## Arguments:

- `_density>=0,0<=_thickness<=50,_max_angle,_opacity,_smoothness>=0`

## Description:

Apply cubism effect on selected images.

## Default values:

`density=50`, `thickness=10`, `max_angle=75`, `opacity=0.7` and `smoothness=0`.

## Example of use:

```
image.jpg cubism ,
```

[0]: 'image.jpg' (640x427x1x3)

# cumulate

Built-in command

## Arguments:

- `{ x | y | z | c }...{ x | y | z | c }` or
- `(no arg)`

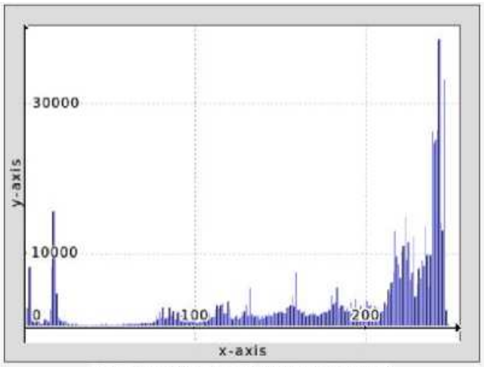
## Description:

Compute the cumulative function of specified image data, optionally along the specified axes.

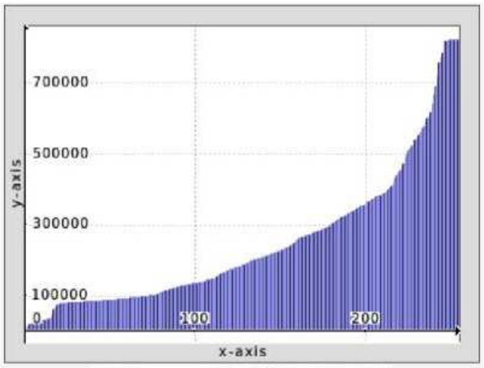
## Example of use:

```
image.jpg +histogram 256 +cumulate[-1] display_graph[-2,-1] 400,300,3
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (400x300x1x3)



[2]: 'image_c2.jpg' (400x300x1x3)

## cup3d

### Arguments:

- `_resolution>0`

### Description:

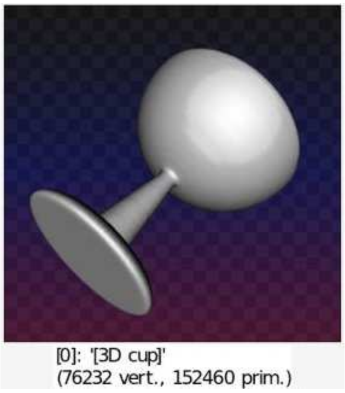
Input 3D cup object.

### Default values:

`resolution=128`.

### Example of use:

```
cup3d ,
```



[0]: '[3D cup]'
(76232 vert., 152460 prim.)

## cursor

**Built-in command**

### Arguments:

- `_mode = { 0:hide | 1:show }`

### Description:

Show or hide mouse cursor for selected instant display windows.

Command selection (if any) stands for instant display window indices instead of image indices.

### Default values:

`mode=1`.

# curvature

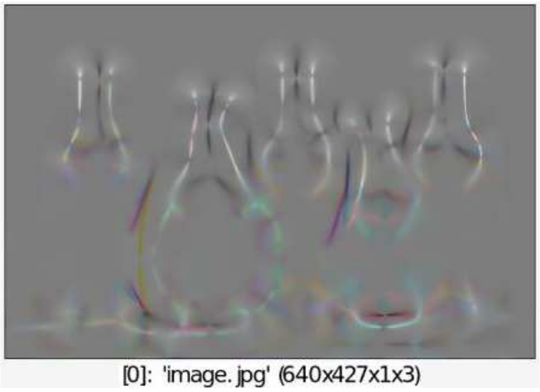
**No arguments**

## Description:

Compute isophote curvatures on selected images.

## Example of use:

```
image.jpg blur 10 curvature
```



[0]: 'image.jpg' (640x427x1x3)

# curve

## Arguments:

- `[xy_coordinates],_thickness>0,_tilt,_tilt_strength[%],_is_closed={ 0:no | 1:yes },_opacity,_color1,...`

## Description:

Draw specified parameterized curve on selected images.

Arguments are:

- `[xy_coordinates]` is the set of XY-coordinates of the curve, specified as a 2-channels image.
- `thickness` is the thickness of the drawing, specified in pixels.
- `tilt` is an angle, specified in degrees.
- `tilt_strength` must be a float value in [0,1] (or in [0,100] if specified as a percentage).
- `is_closed` is a boolean which tells if the curve is closed or not.

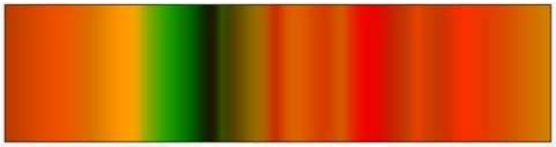
## Default values:

`thickness=0`, `tilt=45`

## Example of use:

```
image.jpg srand 3 16,1,1,4,u s. c,2 rbf[-2,-1] 1000,0,1 n[-2] 10,
{w#0-10} n[-1] 10,{h#0-10} a[-2,-1] c curve[-2] [-1],
6,0,0,0,1,0,128,0
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[u]' (1000x1x1x2)

---

# curve3d

## Arguments:

- `_"x(t)",_"y(t)",_"z(t)",_"r(t)",_resolution>1,_tmin,_tmax,_nb_sides>=0,_is_cl 0 | 1 }`

## Description:

Input 3D curve with specified parameterization.

If `r(t)==0` or `nb_sides<3`, the generated 3D object is composed of segments only.

## Default values:

`x(t)=cos(2*pi*t)`, `y(t)=sin(2*pi*t)`, `z(t)=t`, `r(t)=0.025`, `resolution=128`, `tmin=0`, `tmax=1`, `nb_sides=16` and `is_closed_curve=0`.

## Example of use:

```
curve3d ,
```

[0]: '[3D Curve]'
(2050 vert., 2064 prim.)

# cut

Built-in command

## Arguments:

- `{ value0[%] | [image0] },{ value1[%] | [image1] }` or
- `[image]`

## Description:

Cut values of selected images in specified range.

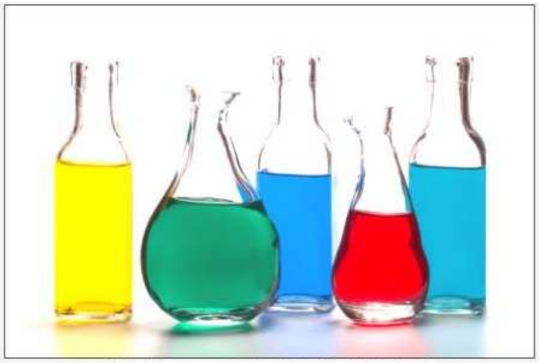
(*equivalent to shortcut command* `c`).

## Examples of use:

### • Example #1

```
image.jpg +add 30% cut[-1] 0,255
```

[0]: 'image.jpg' (640x427x1x3)

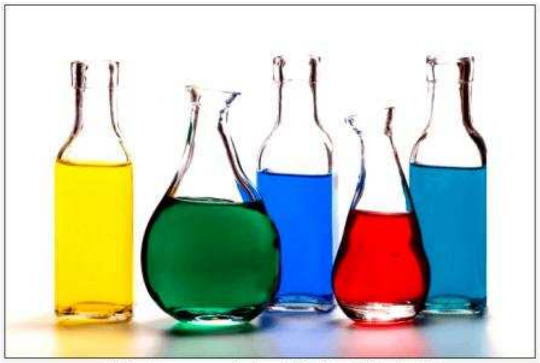


[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +cut 25%,75%
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

# cylinder3d

## Arguments:

- `_radius,_height,_nb_subdivisions>0`

## Description:

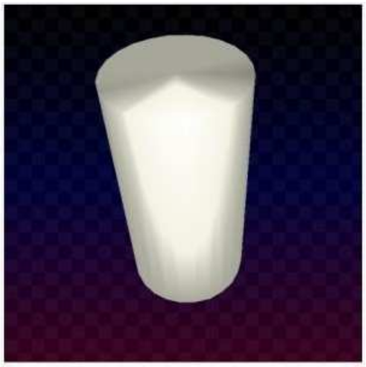
Input 3D cylinder at (0,0,0), with specified geometry.

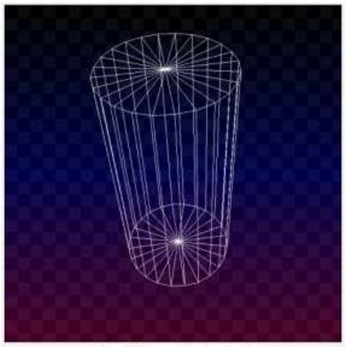
## Default values:

`radius=1`, `height=1` and `nb_subdivisions=24`.

## Example of use:

```
cylinder3d 10,40 +primitives3d 1 color3d[-2] ${-rgb}
```


[0]: '[3D cylinder]' (50 vert., 72 prim.)


[1]: '[3D cylinder]_c1'
(50 vert., 120 prim.)

# da_freeze

**No arguments**

## Description:

Convert each of the selected dynamic arrays into a 1-column image whose height is the number of array elements.

# dct

## Arguments:

- `_{ x | y | z }...{ x | y | z }` or
- `(no arg)`

## Description:

Compute the discrete cosine transform of selected images, optionally along the specified axes only.

Output images are always evenly sized, so this command may change the size of the selected images.

## Default values:

(no arg)

## See also:

**idct** .

This command has a **tutorial page**.

## Example of use:

```
image.jpg +dct +idct[-1] abs[-2] +[-2] 1 log[-2]
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x428x1x3)

[2]: 'image_c2.jpg' (640x428x1x3)

# deblur

## Arguments:

- `amplitude[%]>=0,_nb_iter>=0,_dt>=0,_regul>=0,_regul_type={ 0:Tikhonov | 1:meancurv. | 2:TV }`

## Description:

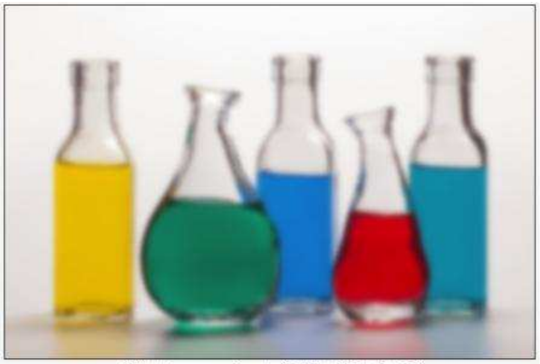
Deblur image using a regularized Jansson-Van Cittert algorithm.

## Default values:

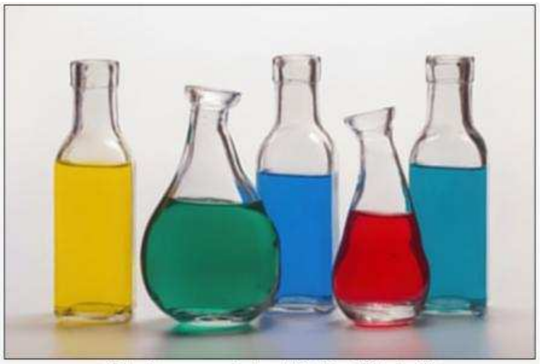
`nb_iter=10`, `dt=20`, `regul=0.7` and `regul_type=1`.

## Example of use:

```
image.jpg blur 3 +deblur 3,40,20,0.01
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# deblur_goldmeinel

## Arguments:

- `sigma>=0,_nb_iter>=0,_acceleration>=0,_kernel_type={ 0:deriche |`

`1:gaussian }.`

## Description:

Deblur selected images using Gold-Meinel algorithm

## Default values:

`nb_iter=8`, `acceleration=1` and `kernel_type=1`.

## Example of use:

```
image.jpg +blur 1 +deblur_goldmeinel[-1] 1
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c3.jpg' (640x427x1x3)

# deblur_richardsonlucy

## Arguments:

- `sigma>=0, nb_iter>=0, _kernel_type={ 0:deriche | 1:gaussian }.`

## Description:

Deblur selected images using Richardson-Lucy algorithm.

## Default values:

`nb_iter=50` and `kernel_type=1`.

## Example of use:

```
image.jpg +blur 1 +deblur_richardsonlucy[-1] 1
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c3.jpg' (640x427x1x3)

# debug

**Built-in command**

### No arguments

## Description:

Activate debug mode.

When activated, the G'MIC interpreter becomes very verbose and outputs additional log messages about its internal state on the standard output (stdout).
This option is useful for developers or to report possible bugs of the interpreter.

# dec

## Arguments:

- `decimal_int1,...`

## Description:

Print specified decimal integers into their binary, octal, hexadecimal and string representations.

# dec2bin

### Arguments:

- `decimal_int1,...`

### Description:

Convert specified decimal integers into their binary representations.

# dec2hex

### Arguments:

- `decimal_int1,...`

### Description:

Convert specified decimal integers into their hexadecimal representations.

# dec2oct

### Arguments:

- `decimal_int1,...`

### Description:

Convert specified decimal integers into their octal representations.

# dec2str

### Arguments:

- `decimal_int1,...`

### Description:

Convert specifial decimal integers into its string representation.

# decompress_clut

## Arguments:

- `_width>0,_height>0,_depth>0`

## Description:

Decompress selected colored keypoints into 3D CLUTs, using a mixed RBF/PDE approach.

## Default values:

`width=height=depth=33` and `reconstruction_colorspace=0`.

---

# decompress_from_keypoints

## Arguments:

- `_width>0,_height>0,_depth>0` or
- `(no arg)`

## Description:

Decompress selected sets of keypoints as images (opt. of specified size).

A set of keypoints is defined as a vector-valued image, such that:

- The first pixel is a vector which encodes the `[ Width,Height,Depth ]` of the decompressed image.
- The second pixel is a vector which encodes `[ Min,Max,Use_RBF ]`, where `Min` and `Max` defines the value range of the decompressed image, and `Use_RBF` tells is the decompression scheme must use RBFs ( `Use_RBF=1` ) or Multiscale Diffusion PDE's ( `Use_RBF=0` ).
- The remaining pixels define the keypoint coordinates and values, as:
  - `[ x_k,y_k,z_k, v1_k,...,vN_k ]` for a 3D target image of N-valued vectors.
  - `[ x_k,y_k, v1_k,...,vN_k ]` for a 2D target image of N-valued vectors.
  - `[ x_k, v1_k,...,vN_k ]` for a 1D target image of N-valued vectors.

where the coordinates `x_k`, `y_k` and `z_k` are defined respectively in ranges `[0,Width-1]`, `[0,Height-1]` and `[0,Depth-1]`.
If the `width`, `height` and `depth` arguments are provided, they define the size of the decompressed image, : overriding then the original image size `[ Width,Height,Depth ]` defined in the keypoints header.

---

# decompress_huffman

## Arguments:

- `[huffman_tree]`

### Description:

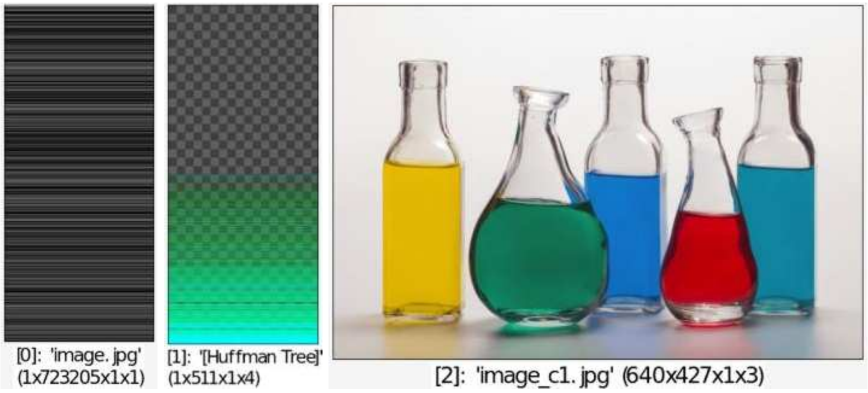
Decompress selected images with Huffman decoding.

### See also:

**compress_huffman**, **huffman_tree**.

### Example of use:

```
image.jpg huffman_tree compress_huffman.. . +decompress_huffman.. .
```


[0]: 'image.jpg' (1x723205x1x1)
[1]: '[Huffman Tree]' (1x511x1x4)
[2]: 'image_c1.jpg' (640x427x1x3)

---

## decompress_rle

**No arguments**

### Description:

Decompress selected data vectors, using RLE algorithm.

---

## deconvolve_fft

### Arguments:

- `[kernel],_regularization>=0`

### Description:

Deconvolve selected images by specified mask in the fourier space.

### Default values:

`regularization>=0`.

### Example of use:

```
image.jpg +gaussian 5 +convolve_fft[0] [1] +deconvolve_fft[-1] [1]
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c1.jpg' (640x427x1x3)

[3]: 'image_c2.jpg' (640x427x1x3)

---

# deform

## Arguments:

- `_amplitude>=0,_interpolation`

## Description:

Apply random smooth deformation on selected images.

`interpolation` can be ***{ 0:none | 1:linear | 2:bicubic }***.

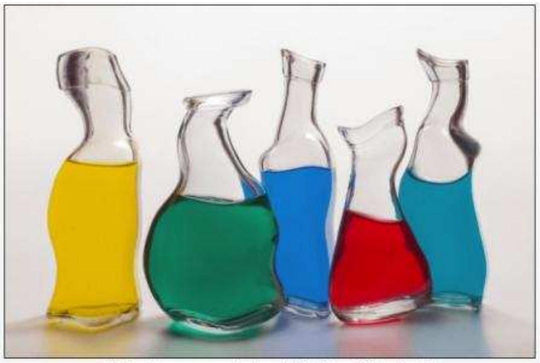
## Default values:

`amplitude=10`.

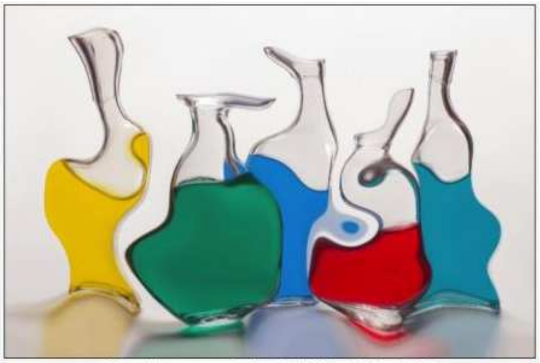
## Example of use:

```
image.jpg +deform[0] 10 +deform[0] 20
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

---

# deg2rad

### No arguments

## Description:

Convert pointwise angle values of selected images, from degrees to radians (apply `i*pi/180`).

---

# deinterlace

## Arguments:

- `_method={ 0 | 1 }`

## Description:

Deinterlace selected images (`method` can be ***{ 0:standard or 1:motion-compensated }***).

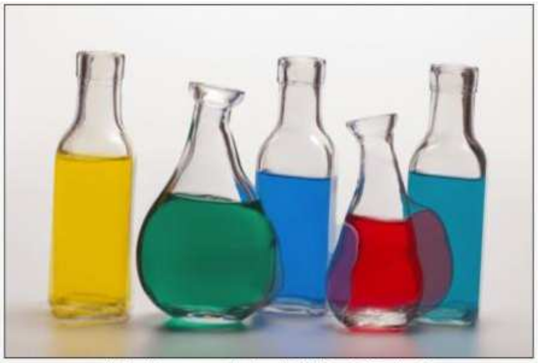
## Default values:

`method=0`.

## Example of use:

```
image.jpg +rotate 3,1,1,50%,50% resize 100%,50% resize 100%,200%,
1,3,4 shift[-1] 0,1 add +deinterlace 1
```

[0]: 'image.jpg' (640x428x1x3)



[1]: 'image_c1.jpg' (640x428x1x3)

# delaunay

## Arguments:

- `_output_type={ 0:image | 1:coordinates/triangles }`

## Description:

Generate discrete 2D Delaunay triangulation of non-zero pixels in selected images.

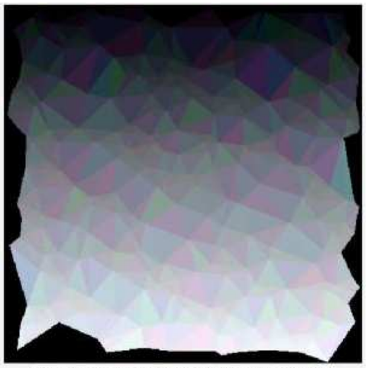
Input images must be scalar.
Each pixel of the output image is a triplet (a,b,c) meaning the pixel belongs to
the Delaunay triangle `ABC` where `a`,`b`,`c` are the labels of the pixels `A`,`B`,`C`.

## Examples of use:

### • Example #1

```
400,400 rand 32,255 100%,100% noise. 0.4,2 eq. 1 mul +delaunay
```

[0]: '[unnamed]' (400x400x1x1)



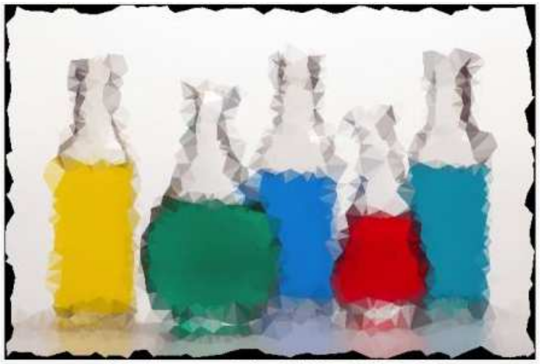
[1]: '[unnamed]_c1' (400x400x1x3)

### • Example #2

```
image.jpg 100%,100% noise. 2,2 eq. 1 delaunay. +blend shapeaverage0
```

[0]: 'image.jpg' (640x427x1x3)

[1]: '[unnamed]' (640x427x1x3)


[2]: 'image_c1.jpg' (640x427x1x3)

## delaunay3d

**No arguments**

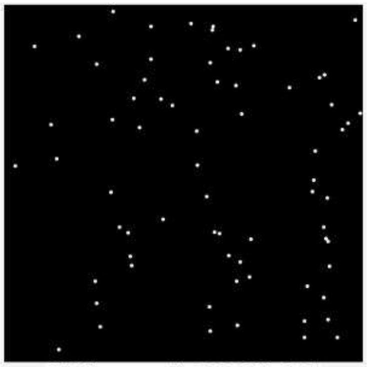
### Description:

Generate 3D Delaunay triangulations from selected images.

One assumes that the selected input images are binary images containing the set of points to mesh. The output 3D object is a mesh composed of non-oriented triangles.

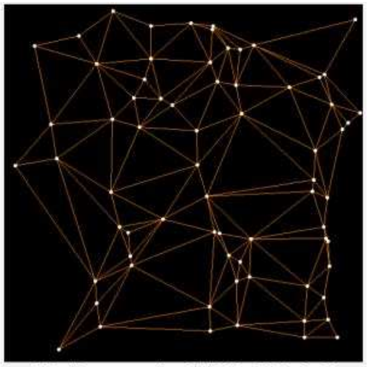
### Example of use:

```
500,500 noise 0.05,2 eq 1 * 255 +delaunay3d color3d[1] 255,128,0
dilate_circ[0] 5 to_rgb[0] +object3d[0] [1],0,0,0,1,1 max[-1] [0]
```



[0]: '[unnamed]' (500x500x1x3)

[1]: '[unnamed]_c1' (70 vert., 111 prim.)



[2]: '[unnamed]_c1' (500x500x1x3)

# delete

Built-in command

## Arguments:

- `filename1[,filename2,...]`

## Description:

Delete specified filenames on disk. Multiple filenames must be separated by commas.

# deltaE

## Arguments:

- `[image],_metric={ 0:deltaE_1976 | 1:deltaE_2000 },"_to_Lab_command"`

## Description:

Compute the CIE DeltaE color difference between selected images and specified [image].

Argument `to_Lab_command` is a command able to convert colors of [image] into a Lab representation.

## Default values:

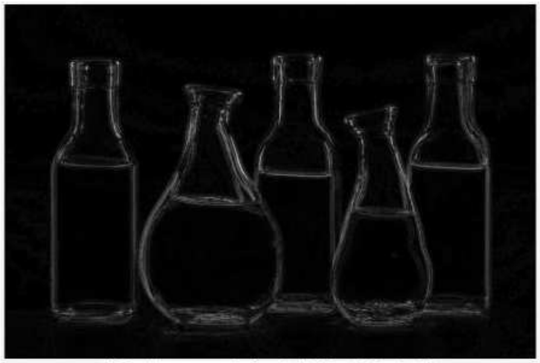
`metric=1` and `to_Lab_command="srgb2lab"`.

## Example of use:

```
image.jpg +blur 2 +deltaE[0] [1],1,srgb2lab
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x1)

# demos

## Arguments:

- `_run_in_parallel={ 0:no | 1:yes | 2:auto }`

## Description:

Show a menu to select and view all G'MIC interactive demos.

# denoise

**Built-in command**

## Arguments:

- `[guide],std_deviation_s[%]>=0,std_deviation_r[%]>=0,_patch_size>0,_lookup_siz` `0 | 1 }` or
- `std_deviation_s[%]>=0,std_deviation_r[%]>=0,_patch_size>0,_lookup_size>0,_smo` `0 | 1 }`

## Description:

Denoise selected images by non-local patch averaging.

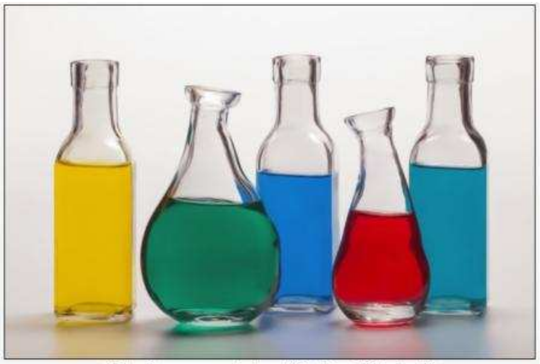
## Default values:

`patch_size=5`, `lookup_size=6` and `smoothness=1`.

## Example of use:

```
image.jpg +denoise 5,5,8
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# denoise_cnn

## Arguments:

- `_noise_type={ 0:soft | 1:heavy | 2:heavy (faster) | 3:poisson+gaussian | 4:poisson+gaussian2 },_patch_size>0`

## Description:

Denoise selected images using a convolutional neural network (CNN).

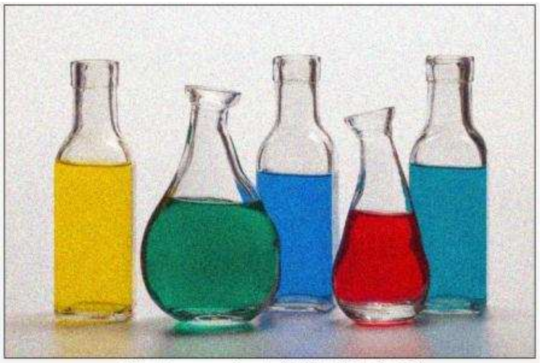
Input value range should be [0,255]. Output value range is [0,255].

## Default values:

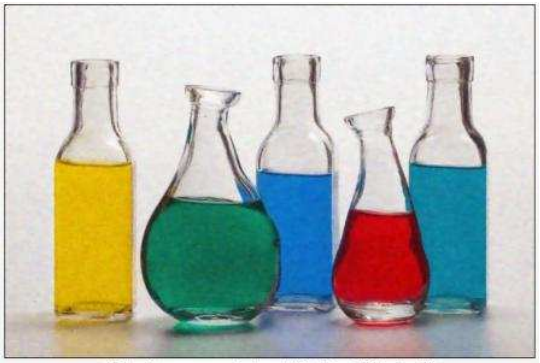
`patch_size=64`.

## Example of use:

```
image.jpg noise 20 cut 0,255 +denoise_cnn
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# denoise_haar

## Arguments:

- `_threshold>=0,_nb_scales>=0,_cycle_spinning>0`

## Description:

Denoise selected images using haar-wavelet thresholding with cycle spinning.

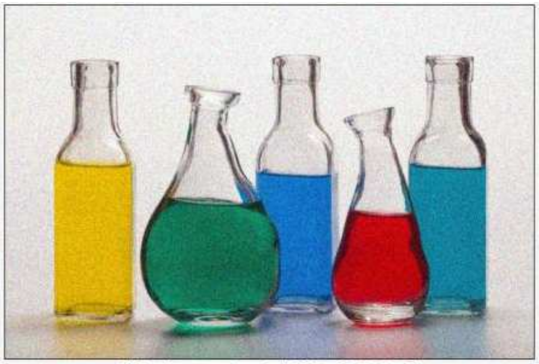
Set `nb_scales==0` to automatically determine the optimal number of scales.

## Default values:

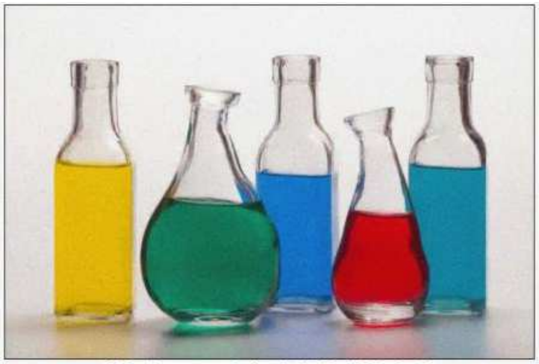
`threshold=1.4`, `nb_scale=0` and `cycle_spinning=10`.

## Example of use:

```
image.jpg noise 20 cut 0,255 +denoise_haar[-1] 0.8
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# denoise_patchpca

## Arguments:

- `_strength>=0,_patch_size>0,_lookup_size>0,_spatial_sampling>0`

## Description:

Denoise selected images using the patch-pca algorithm.

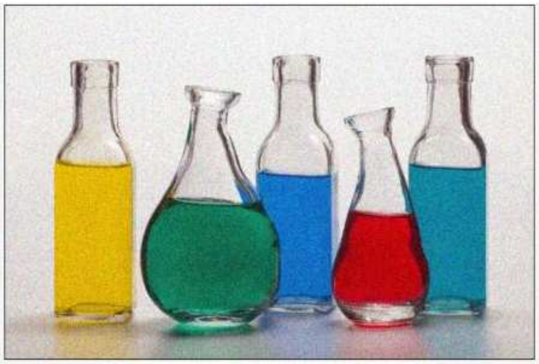
## Default values:

`patch_size=7`, `lookup_size=11`, `details=1.8` and `spatial_sampling=5`.

## Example of use:

```
image.jpg +noise 20 cut[-1] 0,255 +denoise_patchpca[-1] ,
```

[0]: 'image. jpg' (640x427x1x3)



[1]: 'image_c1. jpg' (640x427x1x3)

[2]: 'image_c2. jpg' (640x427x1x3)

## deriche

Built-in command

### Arguments:

- `std_deviation>=0[%],order={ 0 | 1 | 2 },axis={ x | y | z | c },_boundary_conditions`

### Description:

Apply Deriche recursive filter on selected images, along specified axis and with

specified standard deviation, order and boundary conditions.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

### Default values:

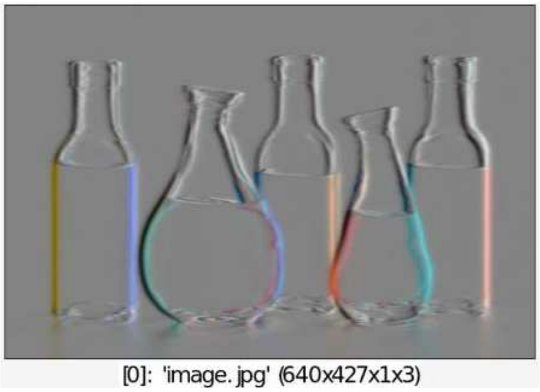
`boundary_conditions=1`.

This command has a **tutorial page**.

### Examples of use:

- **Example #1**

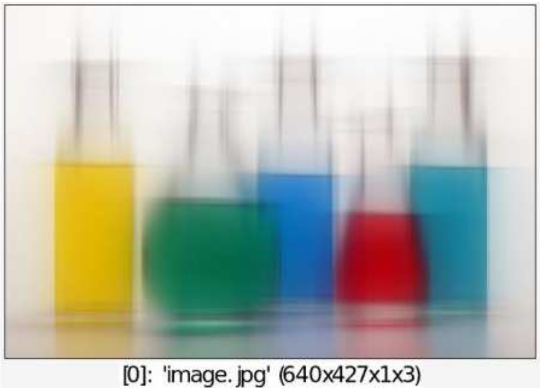
```
image.jpg deriche 3,1,x
```



[0]: 'image.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg +deriche 30,0,x deriche[-2] 30,0,y add
```



[0]: 'image.jpg' (640x427x1x3)

# detect_skin

## Arguments:

- `0<=tolerance<=1,_skin_x,_skin_y,_skin_radius>=0`

## Description:

Detect skin in selected color images and output an appartenance probability map.

Detection is performed using CbCr chromaticity data of skin pixels.
If arguments `skin_x`, `skin_y` and `skin_radius` are provided, skin pixels are learnt
from the sample pixels inside the circle located at ( `skin_x` , `skin_y` ) with radius `skin_radius` .

## Default values:

`tolerance=0.5` and `skin_x=skiny=radius=-1`.

# diagonal

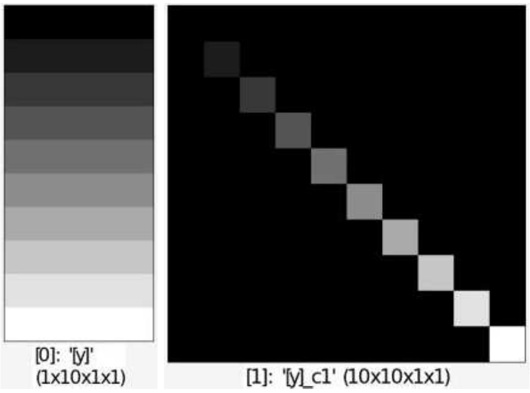
**No arguments**

## Description:

Transform selected vectors as diagonal matrices.

## Example of use:

```
1,10,1,1,'y' +diagonal
```



# diffusiontensors

## Arguments:

- `_sharpness>=0,0<=_anisotropy<=1,_alpha[%],_sigma[%],is_sqrt={ 0 | 1 }`

## Description:

Compute the diffusion tensors of selected images for edge-preserving smoothing algorithms.

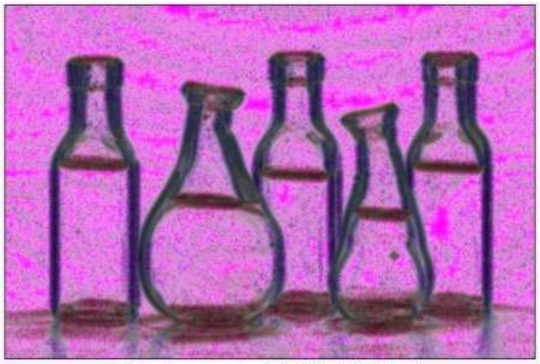
## Default values:

`sharpness=0.7`, `anisotropy=0.3`, `alpha=0.6`, `sigma=1.1` and `is_sqrt=0`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg diffusiontensors 0.8 abs pow 0.2
```



[0]: 'image.jpg' (640x427x1x3)

## dijkstra

### Arguments:

- `starting_vertex>=0,_ending_vertex={ -1:none | >=0 }`

### Description:

Compute minimal distances/paths in selected graphs, from specified `starting_vertex` to all other vertices (opt. only until `ending_vertex` has been reached).

A graph of `N` vertices is specified as a `NxN` adjacency matrix giving the weights of all edges connecting vertices (set to `inf` when two vertices are not connected).
This command return a `1xNx1x2` image containing the `[distance,parent]` information :

- `distance` is the minimal distance from vertex `#y` to the `starting_vertex` (i.e. the sum of edge weights composing the minimal path between these two vertices).
- `parent` is the index of the next vertex that must be followed to reaches the `starting_vertex` through the minimal path.

### Default values:

`ending_vertex=-1`

## dilate

**Built-in command**

### Arguments:

- `size>=0` or
- `size_x>=0,size_y>=0,size_z>=0` or
- `[kernel],_boundary_conditions,_is_real={ 0:binary-mode | 1:real-mode }`

### Description:

Dilate selected images by a rectangular or the specified structuring element.

`boundary_conditions` can be *`{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`*.

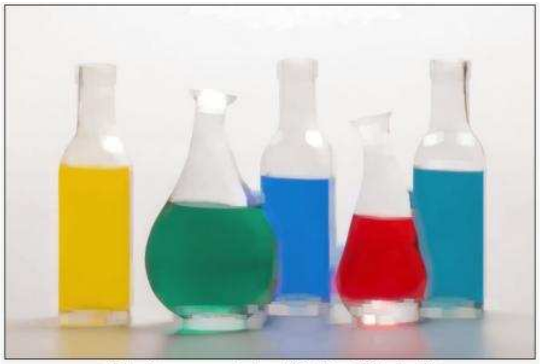
## Default values:

`size_z=1`, `boundary_conditions=1` and `is_real=0`.

## Example of use:

```
image.jpg +dilate 10
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# dilate_circ

## Arguments:

- `_size>=0,_boundary_conditions,_is_real={ 0 | 1 }`

## Description:

Apply circular dilation of selected images by specified size.

`boundary_conditions` can be *`{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`*.

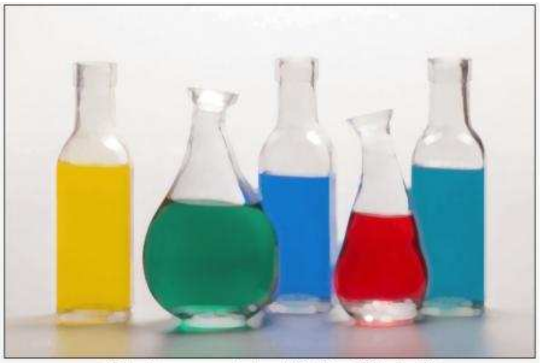
## Default values:

`boundary_conditions=1` and `is_real=0`.

## Example of use:

```
image.jpg +dilate_circ 7
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# dilate_oct

## Arguments:

- `_size>=0,_boundary_conditions,_is_real={ 0 | 1 }`

## Description:

Apply octagonal dilation of selected images by specified size.

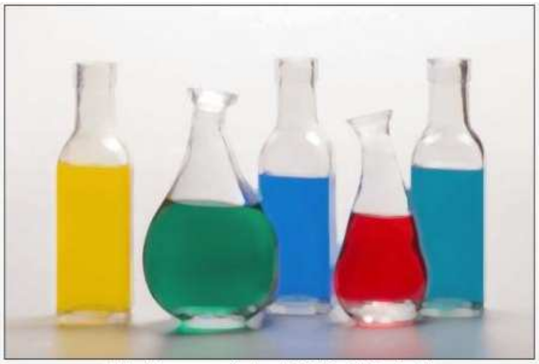
## Default values:

`boundary_conditions=1` and `is_real=0`.

## Example of use:

```
image.jpg +dilate_oct 7
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# dilate_threshold

## Arguments:

- `size_x>=1,size_y>=1,size_z>=1,_threshold>=0,_boundary_conditions`

### Description:

Dilate selected images in the (X,Y,Z,I) space.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

### Default values:

`size_y=size_x`, `size_z=1`, `threshold=255` and `boundary_conditions=1`.

## direction2rgb

**No arguments**

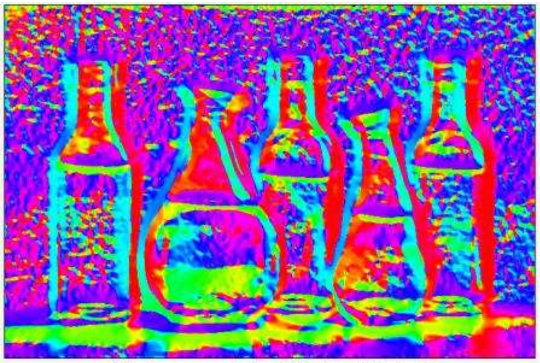
### Description:

Compute RGB representation of selected 2D direction fields.

### Example of use:

```
image.jpg luminance gradient append c blur 2 orientation
+direction2rgb
```

[0]: 'image.jpg' (640x427x1x2)



[1]: 'image_c1.jpg' (640x427x1x3)

## discard

**Built-in command**

### Arguments:

- `_value1,_value2,...` or
- `{ x | y | z | c}...{ x | y | z | c},_value1,_value2,...` or
- `(no arg)`

### Description:

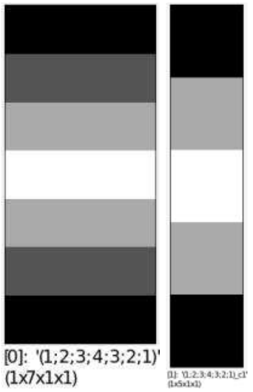
Discard specified values in selected images or discard neighboring duplicate values,

optionally only for the values along the first of a specified axis.
If no arguments are specified, neighboring duplicate values are discarded.
If all pixels of a selected image are discarded, an empty image is returned.

## Examples of use:

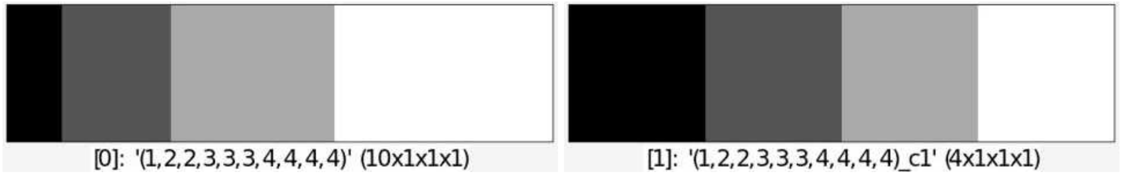
### • Example #1

```
(1;2;3;4;3;2;1) +discard 2
```



### • Example #2

```
(1,2,2,3,3,3,4,4,4,4) +discard x
```


[0]: '(1,2,2,3,3,3,4,4,4,4)' (10x1x1x1)
[1]: '(1,2,2,3,3,3,4,4,4,4)_c1' (4x1x1x1)

# displacement

Built-in command

## Arguments:

- `[source_image],_smoothness,_precision>=0,_nb_scales>=0,_iteration_max>=0,is_b`
`0 | 1 },_[guide]`

## Description:

Estimate displacement field between specified source and selected target images.

If `smoothness>=0`, regularization type is set to isotropic, else to anisotropic.
If `nbscales==0`, the number of scales used is estimated from the image size.

## Default values:

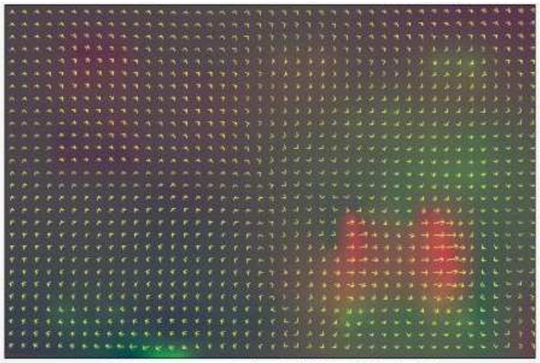
`smoothness=0.1`, `precision=5`, `nb_scales=0`, `iteration_max=10000`, `is_backward=1` and `[guide]=(unused)`.

## Example of use:

```
image.jpg +rotate 3,1,0,50%,50% +displacement[-1] [-2] quiver[-1]
[-1],15,1,1,1,{1.5*iM}
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)


[2]: 'image_c2.jpg' (640x427x1x2)

# display

### No arguments

## Description:

Display selected images in an interactive window.

(*equivalent to shortcut command* `d`).

When invoked with a `+` prefix (i.e. `+display`), the command outputs its log messages on `stdout` rather than on `stderr`.
Display window #0 is used as the default window for the display, if already opened.

Available controls are shown below (where `LMB` = Left mouse button, `RMB` = Right mouse button, `MMB` = Middle mouse button and `MW` = Mouse wheel).

- **Thumbnail navigation bar:**

`TAB` : Show/hide thumbnails - `LMB` : Select thumbnail or shift thumbnail bar - `0` - `9` , `ARROWS` (opt. `+SHIFT` ), `B` , `BACKSPACE` , `C` , `E` , `END` , `H` , `HOME` , `SPACE` : Navigate and select thumbnails (add `CTRL` if mouse pointer is outside thumbnail bar).

- **Image view:**

`LMB` or `MMB` : Image pan - `RMB` or `MW` : Image zoom - `ARROWS` (opt. `+SHIFT` ), `HOME` , `END` : Shift view - `A` : Switch alpha rendering - `C` : Center view - `E` : Go to lower-right corner - `ENTER` : Reset view - `G` : Toggle grid - `H` : Go to upper-left corner - `K` : Switch background - `M` : Toggle 3D view - `N` : Switch normalization - `P` : Print info about current image pixel on `stdout` - `PAGEUP` or `PAGEDOWN` : Raise/lower base channel - `R` : Rotate image - `V` : Crop image - `Z` : Switch zoom factor - `0` - `9` : Set zoom factor.

- **3D mesh view:**

`LMB` : Mesh rotation - `CTRL+LMB` or `MMB` : Mesh pan - `RMB` : Mesh zoom - `A` : Toggle axes - `D` : Switch face side mode - `F` : Change focale - `J` : Start/stop animation - `K` : Switch background - `O` : Switch outline mode - `P` : Print 3D pose matrix on `stdout` - `R` : Switch rendering mode - `T` : Switch motion rendering mode - `X` : Show/hide bounding-box - `U` : Switch animation mode - `Z` : Toggle z-buffer.

- **2D images specific:**

`CTRL+LMB` : Rectangular selection.

- **3D volumetric images specific:**

`CTRL+MW` : Pan along orthogonal axis - `X` : Reset area layout.

- **Window size, decoration and data I/O:**

`CTRL+C` : Decrease window size - `CTRL+D` : Increase window size - `CTRL+F` : Toggle fullscreen - `CTRL+I` : Toggle info label - `CTRL+O` : Save copy of image as a `.gmz` file - `CTRL+L` : Save copy of image list as `.gmz` file - `CTRL+S` : Save screenshot as a `.png` file - `CTRL+W` : Start/stop window recording - `CTRL+X` : Toggle cursor.

- **Configuration variables:**

The viewer configuration can be tuned by assigning the following variables:

  - `_display_selected` is an integer or an image name that tells which image is selected by default.
  - `_display_alpha` can be *`{ 0:off | 1:on | 2:over black | 3:over gray | 4:over white }`* (default value: `0` ).
  - `_display_background` , an integer in range [ 0,9 ] (default value: `3` ).
  - `_display_cursor` can be *`{ 0:off | 1:on (2D only) | 2:on (+3D volumetric images) }`* (default value: `1` ).
  - `_display_is_grid` can be *`{ 0:off | 1:on }`* (default value: `1` ).
  - `_display_is_info` can be *`{ 0:off | 1:on }`* (default value: `1` ).
  - `_display_normalization` can be *`{ -1:auto | 0:off | 1:cut | 2:stretch channelwise | 3:stretch global | 4: stretch (global once) }`* (default value: `-1` ).
  - `_display_print_images` can be *`{ 0:off | N>0 }`* (default value: `5` ). It sets the max

number `N` of images whose information is initially printed on `stderr` or `stdout`.

  - `_display_3d_is_rendered` can be *`{ 0:off | 1:on }`* (default value: `1`).
  - `_display_3d_rendering_mode` can be *`{ 0:dots | 1:wireframe | 2:flat | 3:flat-shaded | 4:gouraud-shaded | 5=phong-shaded }`* (default value: `4`).
  - `_display_3d_outline_mode` can be *`{ 0:no-outline | 1:black-outline | 2:gray-outline | 3:red-outline | 4:green-outline | 5:blue-outline | 6:white-outline }`* (default value: `0`).
  - `_display_3d_motion_rendering_mode` can be *`{ -1:bounding-box | 0:dots | 1:wireframe | 2:flat | 3:flat-shaded | 4:gouraud-shaded | 5=phong-shaded }`* (default value: `3`).
  - `_display_3d_motion_time_limit` is specified in ms. Above this time, motion rendering toggle to `bounding-box` mode (default value: `300`).
  - `_display_3d_side_mode` can be *`{ 0:single-sided | 1:double-sided | 2:single-sided (flipped) }`* (default value: `0`).
  - `_display_3d_is_zbuffer` can be *`{ 0:off | 1:on }`* (default value: `1`).
  - `_display_3d_focale` can be *`{ <0: perspective projection w/o sprite zooming, 0: parallel projection | >0: perspective projection }`* (default value: 1.5).
  - `_display_3d_is_axes` can be *`{ 0:off | 1:on }`* (default value: `1`).
  - `_display_3d_is_bounding_box` can be *`{ 0:off | 1:on }`* (default value: `0`).
  - `_display_3d_background` is an unsigned integer in range [0,11] (default value: `11`).
  - `_display_3d_pose` is a sequence of 12 values that defines the current 3D pose matrix (read/write).
  - `_display_3d_animation` can be *`{ 0:off | 1:forward | 2:backward }`* (default value: `0`).
  - `_display_3d_animation_mode` can be *`{ 0-3:X-axis | 4-7:Y-axis | 8-11:Z-axis | 12-15:XYZ-axes }`* (default value: `4`).

# display0

### No arguments

## Description:

Display selected images in an interactive window, without normalization and alpha mode activated.

# display_array

## Arguments:

- `_width>0,_height>0`

## Description:

Display images in interactive windows where pixel neighborhoods can be explored.

## Default values:

`width=13` and `height=width`.

---

# display_camera

**No arguments**

## Description:

Open camera viewer.

This command requires features from the OpenCV library (not enabled in G'MIC by default).

---

# display_clut

## Arguments:

- `_image_resolution>0,_clut_resolution>0`

## Description:

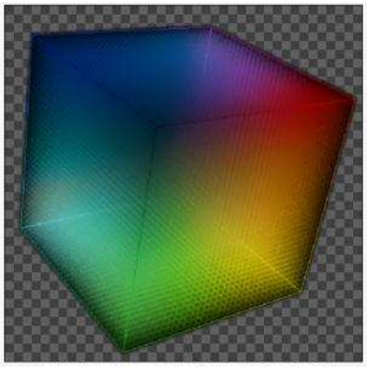
Display selected 3D color LUTs.

## Default values:

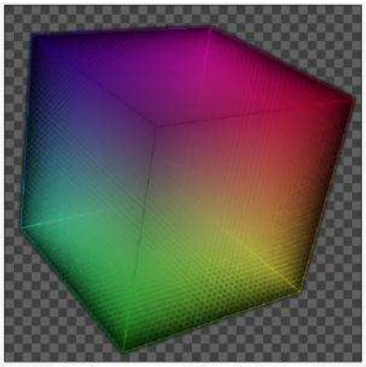
`image_resolution=320` and `clut_resolution=33`.

## Example of use:

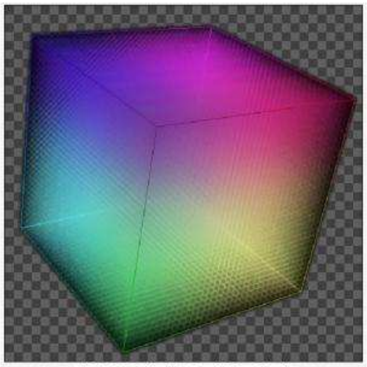
```
clut tealorange clut summer clut 60s display_clut 400
```



[0]: 'tealorange' (400x400x1x4)



[1]: 'summer' (400x400x1x4)



[2]: '60s' (400x400x1x4)

# display_fft

**No arguments**

## Description:

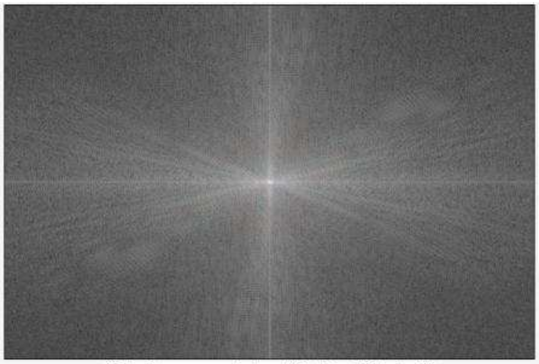
Display fourier transform of selected images, with centered log-module and argument.

(*equivalent to shortcut command* `dfft`).

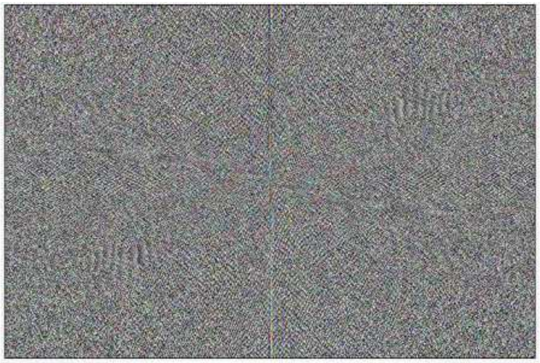
## Example of use:

```
image.jpg +display_fft
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c2.jpg' (640x427x1x3)

# display_graph

## Arguments:

- `_width>=0,_height>=0,_plot_type,_vertex_type,_xmin,_xmax,_ymin,_ymax,_xlabel,`

## Description:

Render graph plot from selected image data.

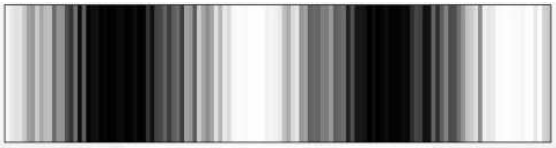
`plot_type` can be *`{ 0:none | 1:lines | 2:splines | 3:bar }`*.
`vertex_type` can be *`{ 0:none | 1:points | 2,3:crosses | 4,5:circles | 6,7:squares }`*.
`xmin`, `xmax`, `ymin`, `ymax` set the coordinates of the displayed xy-axes.
if specified `width` or `height` is `0`, then image size is set to half the screen size.

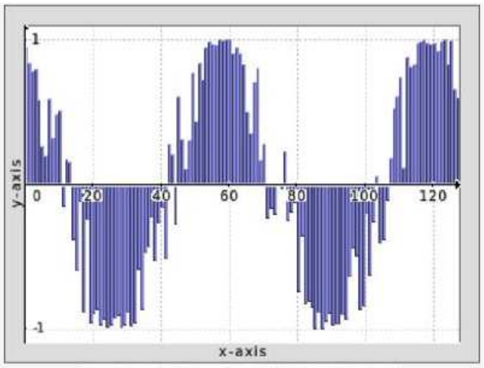
## Default values:

`width=0`, `height=0`, `plot_type=1`, `vertex_type=1`, 'xmin=xmax=ymin=ymax=0 (auto)', `xlabel="x-axis"` and `ylabel="y-axis"`.

## Example of use:

```
128,1,1,1,'cos(x/10+u)' +display_graph 400,300,3
```


[0]: '[cos(x/10+u)]' (128x1x1x1)



[1]: '[cos(x/10+u)]_c1' (400x300x1x3)

# display_histogram

## Arguments:

- `_width>=0,_height>=0,_clusters>0,_min_value[%],_max_value[%],_show_axes={ 0 | 1 },_expression.`

## Description:

Render a channel-by-channel histogram.

If selected images have several slices, the rendering is performed for all input slices.
`expression` is a mathematical expression used to transform the histogram data for visualization purpose.

(*equivalent to shortcut command* `dh`).

if specified `width` or `height` is `0`, then image size is set to half the screen size.

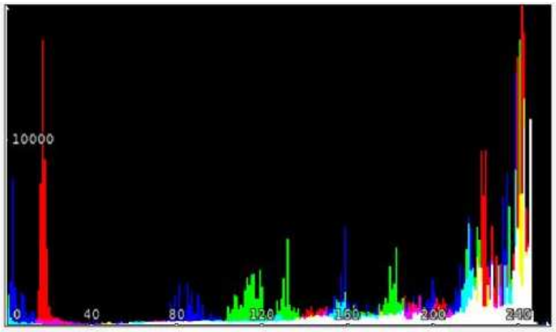
## Default values:

`width=0`, `height=0`, `clusters=256`, `min_value=0%`, `max_value=100%`, `show_axes=1` and `expression=i`.

## Example of use:

```
image.jpg +display_histogram 512,300
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (512x300x1x3)

# display_parametric

## Arguments:

- `_width>0,_height>0,_outline_opacity,_vertex_radius>=0,_is_antialiased={ 0 | 1 },_is_decorated={ 0 | 1 },_xlabel,_ylabel`

## Description:

Render 2D or 3D parametric curve or point clouds from selected image data.

Curve points are defined as pixels of a 2 or 3-channel image.
If the point image contains more than 3 channels, additional channels define the (R,G,B) color for each vertex.
If `outline_opacity>1`, the outline is colored according to the specified vertex colors and `outline_opacity-1` is used as the actual drawing opacity.

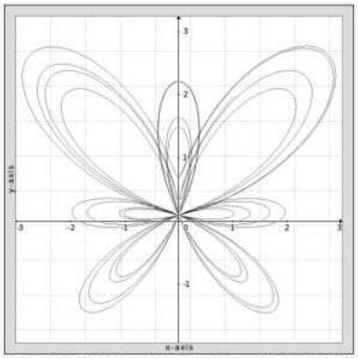
## Default values:

`width=512`, `height=width`, `outline_opacity=3`, `vertex_radius=0`, `is_antialiased=1`, `is_decorated=1`, `xlabel="x-axis"` and `ylabel="y-axis"`.

## Examples of use:

### • Example #1

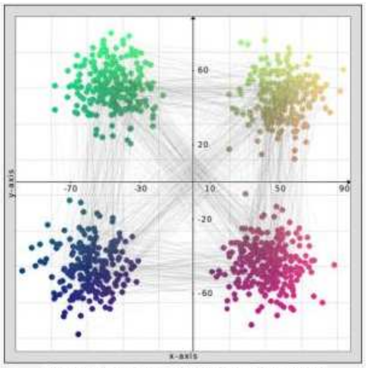
```
1024,1,1,2,'t=x/40;(!c?sin(t):cos(t))*(exp(cos(t))-2*cos(4*t)-sin(t/
12)^5)' display_parametric 512,512
```



[0]: '[t=x/40;(!c?sin(t):cos(t))*(exp(cos(t))-2*cos(4*t)-sin(t/12)^5)]'
(512x512x1x1)

- **Example #2**

```
1000,1,1,2,u(-100,100) quantize 4,1 noise 12 channels 0,2 +normalize
0,255 append c display_parametric 512,512,0.1,8
```



[0]: '[u(-100,100)]' (512x512x1x3)

---

# display_polar

## Arguments:

- `_width>32,_height>32,_outline_type,_fill_R,_fill_G,_fill_B,_theta_start,_thet`

## Description:

Render polar curve from selected image data.

`outline_type` can be *`{ r<0:dots with radius -r | 0:no outline | r>0:lines+dots with radius r }`*.
`fill_color` can be *`{ -1:no fill | R,G,B:fill with specified color }`*.

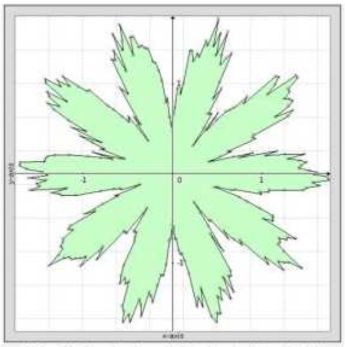
## Default values:

`width=500`, `height=width`, `outline_type=1`, `fill_R=fill_G=fill_B=200`, `theta_start=0`, `theta_end=360`, `xlabel="x-axis"` and `ylabel="y-axis"`.

## Examples of use:

- **Example #1**

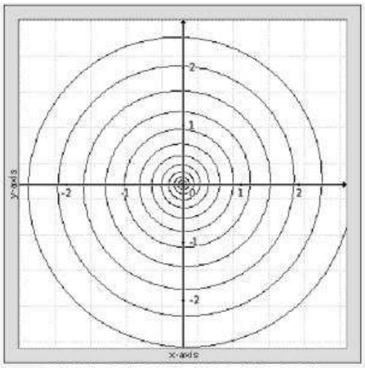
```
300,1,1,1,'0.3+abs(cos(10*pi*x/w))+u(0.4)' display_polar
512,512,4,200,255,200
```



[0]: '[0.3+abs(cos(10*pi*x/w))+u(0.4)]' (512x512x1x3)

- **Example #2**

```
3000,1,1,1,'x^3/1e10' display_polar 400,400,1,-1,,,0,{15*360}
```



[0]: '[x^3/1e10]' (400x400x1x3)

# display_quiver

## Arguments:

- `_size_factor>0,_arrow_size>=0,_color_mode={ 0:monochrome | 1:grayscale | 2:color }`

## Description:

Render selected images of 2D vectors as a field of 2D arrows.

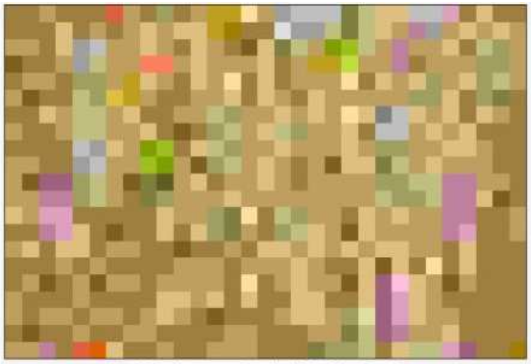
(*equivalent to shortcut command* `dq`).

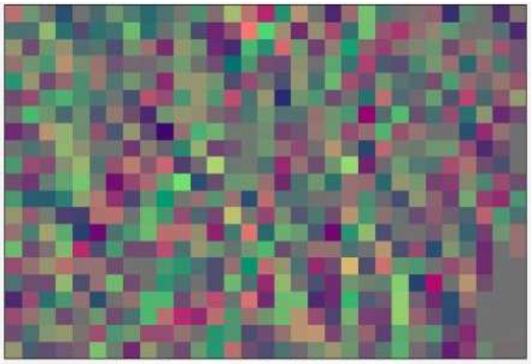
## Default values:

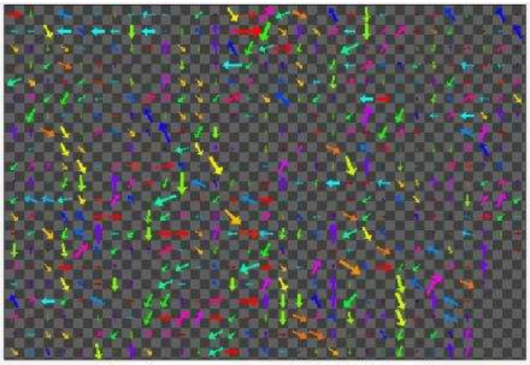
`size_factor=16`, `arrow_size=1.5` and `color_mode=1`.

## Example of use:

```
image.jpg +luminance gradient[-1] xy rv[-2,-1] *[-2] -1 a[-2,-1] c
crop 60,10,90,30 +display_quiver[1] ,
```


[0]: 'image.jpg' (31x21x1x3)


[1]: 'image_c1.jpg' (31x21x1x2)


[2]: '[unnamed]_c1' (496x336x1x4)

---

# display_rgba

## Arguments:

- `_background_RGB_color`

## Description:

Render selected RGBA images over a checkerboard or colored background.

(*equivalent to shortcut command* `drgba`).

## Default values:

`background_RGB_color=undefined` (checkerboard).

## Example of use:

```
image.jpg +norm threshold[-1] 40% blur[-1] 3 normalize[-1] 0,255
append c display_rgba
```

[0]: 'image.jpg' (640x427x1x3)

---

# display_tensors

## Arguments:

- `_size_factor>0,_ellipse_size>=0,_color_mode={ 0:monochrome | 1:grayscale | 2:color },_outline>=0`

## Description:

Render selected images of tensors as a field of 2D ellipses.

(*equivalent to shortcut command* `dt`).

## Default values:

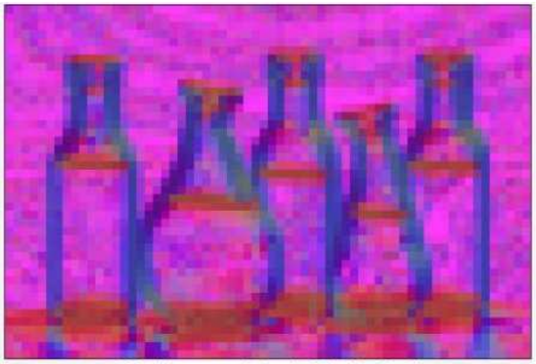
`size_factor=16`, `ellipse_size=1.5`, `color_mode=2` and `outline=2`.

This command has a **tutorial page**.

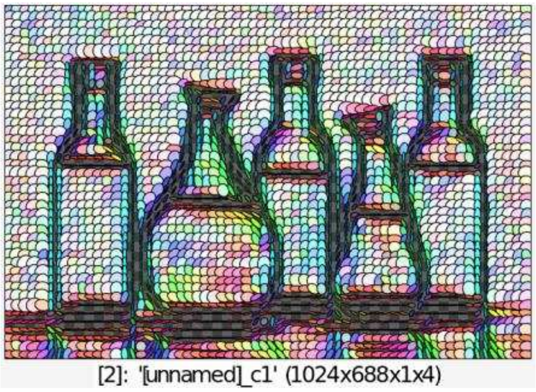
## Example of use:

```
image.jpg +diffusiontensors 0.1,0.9 rescale2d. 64 +display_tensors. 16,2
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (64x43x1x3)



[2]: '[unnamed]_c1' (1024x688x1x4)

# display_voxels3d

**No arguments**

## Description:

Display selected images as set of 3D voxels.

(*equivalent to shortcut command* `dv3d`).

# display_warp

## Arguments:

- `_cell_size>0`

## Description:

Render selected 2D warping fields.

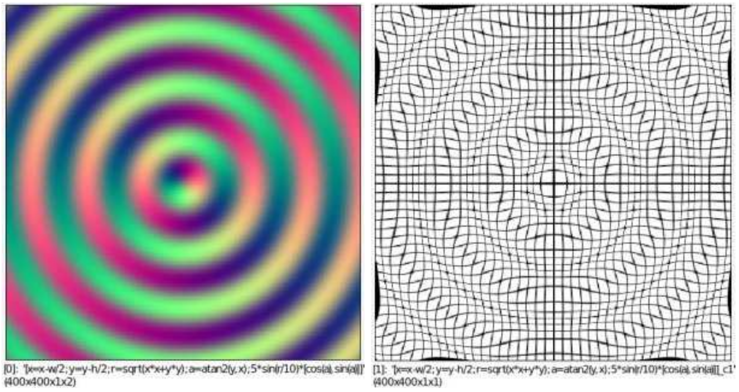
(*equivalent to shortcut command* `dw`).

## Default values:

`cell_size=15`.

## Example of use:

```
400,400,1,2,'x=x-w/2;y=y-h/2;r=sqrt(x*x+y*y);a=atan2(y,x);5*sin(r/
10)*[cos(a),sin(a)]' +display_warp 10
```


[0]: '[x=x-w/2;y=y-h/2;r=sqrt(x*x+y*y);a=atan2(y,x);5*sin(r/10)*[cos(a),sin(a)]]'
(400x400x1x2)
[1]: '[x=x-w/2;y=y-h/2;r=sqrt(x*x+y*y);a=atan2(y,x);5*sin(r/10)*[cos(a),sin(a)]]_c1'
(400x400x1x1)

# distance

Built-in command

## Arguments:

- `isovalue[%],_metric` or
- `isovalue[%],[metric],_method`

## Description:

Compute the unsigned distance function to specified isovalue, opt. according to a custom metric.

`metric` can be ***{ 0:chebyshev | 1:manhattan | 2:euclidean | 3:squared-euclidean }***.
`method` can be ***{ 0:fast-marching | 1:low-connectivity dijkstra | 2:high-connectivity dijkstra | 3:1+return path | 4:2+return path }***.

## Default values:

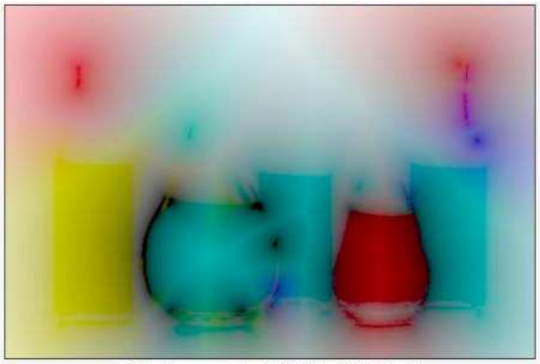
`metric=2` and `method=0`.

This command has a **tutorial page**.

## Examples of use:

### • Example #1

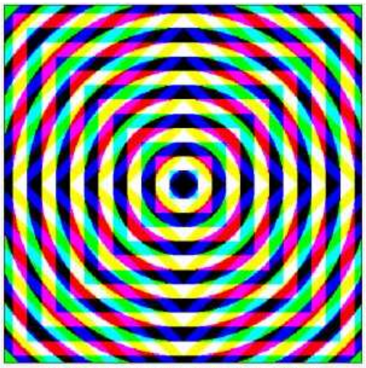
```
image.jpg threshold 20% distance 0 pow 0.3
```



[0]: 'image.jpg' (640x427x1x3)

- **Example #2**

```
400,400 set 1,50%,50% +distance[0] 1,2 +distance[0] 1,1 distance[0]
1,0 mod 32 threshold 16 append c
```



[0]: '[unnamed]' (400x400x1x3)

## distribution3d

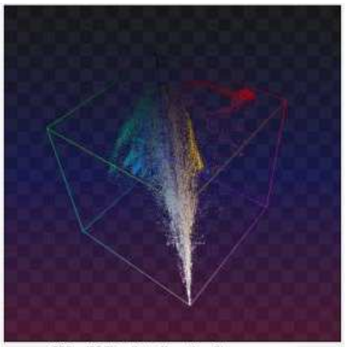
**No arguments**

### Description:

Get 3D color distribution of selected images.

### Example of use:

```
image.jpg distribution3d colorcube3d primitives3d[-1] 1 add3d
```



[0]: '[3D distribution]'
(273288 vert., 273292 prim.)

---

# ditheredbw

**No arguments**

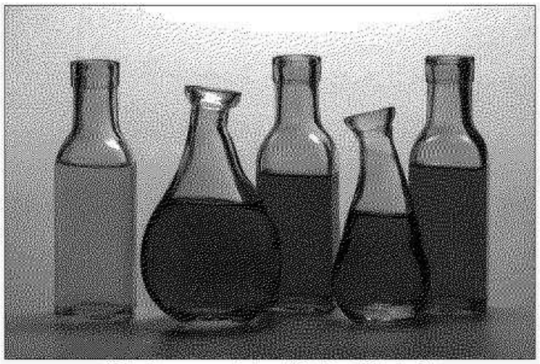
## Description:

Create dithered B&W version of selected images.

## Example of use:

```
image.jpg +equalize ditheredbw[-1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x1)

---

# div

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

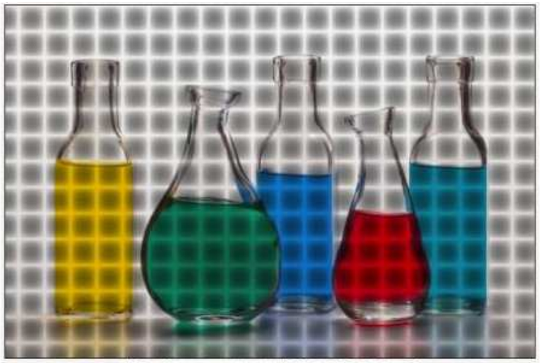
Divide selected images by specified value, image or mathematical expression, or compute the pointwise quotient of selected images.

(*equivalent to shortcut command* `/`).

## Examples of use:

### • Example #1

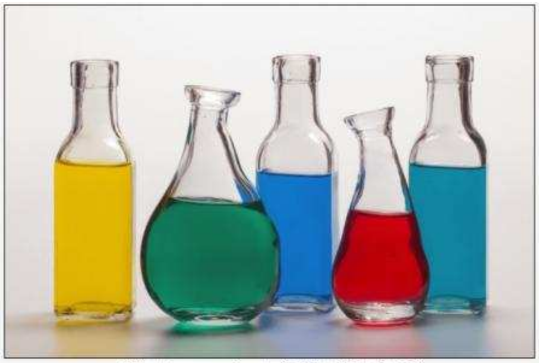
```
image.jpg div '1+abs(cos(x/10)*sin(y/10))'
```



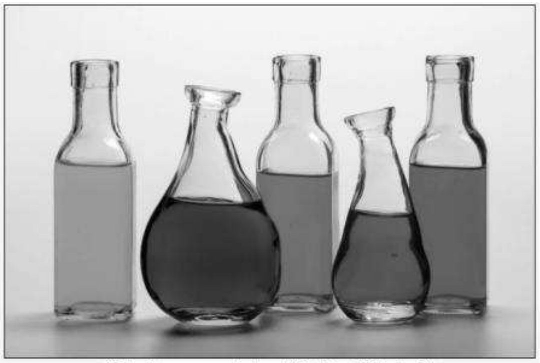
[0]: 'image.jpg' (640x427x1x3)

### • Example #2

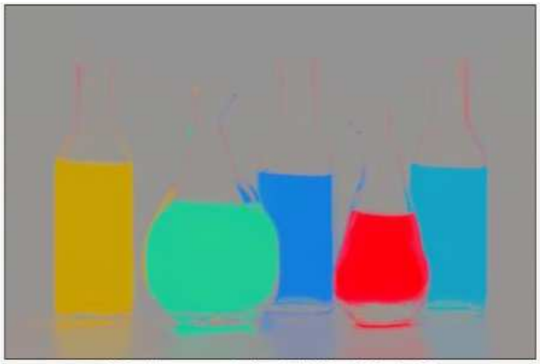
```
image.jpg +norm add[-1] 1 +div
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

# div3d

Built-in command

## Arguments:

- `factor` or
- `factor_x,factor_y,_factor_z`

## Description:

Scale selected 3D objects isotropically or anisotropically, with the inverse of specified

factors.

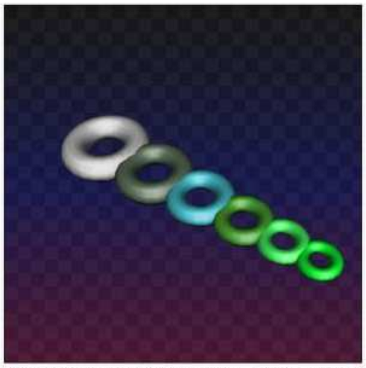
(*equivalent to shortcut command* `/3d`).

## Default values:

`factor_z=1`.

## Example of use:

```
torus3d 5,2 repeat 5 { +add3d[-1] 12,0,0 div3d[-1] 1.2 color3d[-1] $
{-rgb} } add3d
```



[0]: '[3D torus]' (1728 vert., 1728 prim.)

# div_complex

## Arguments:

- `[divider_real,divider_imag],_epsilon>=0`

## Description:

Perform division of the selected complex pairs (real1,imag1,...,realN,imagN) of images by

specified complex pair of images (divider_real,divider_imag).
In complex pairs, the real image must be always located before the imaginary image in the image list.

## Default values:

`epsilon=1e-8`.

# divergence

### No arguments

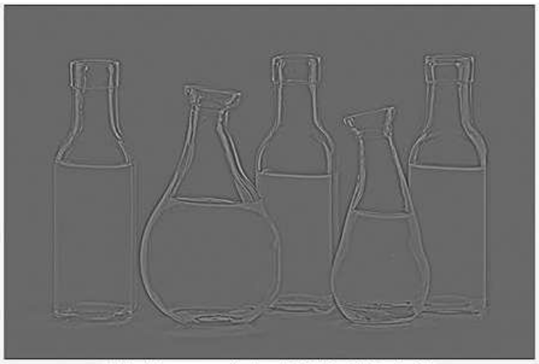
## Description:

Compute divergence of selected vector fields.

## Example of use:

```
image.jpg luminance +gradient append[-2,-1] c divergence[-1]
```

[0]: 'image.jpg' (640x427x1x1)


[1]: 'image_c1.jpg' (640x427x1x1)

# do

**Built-in command**

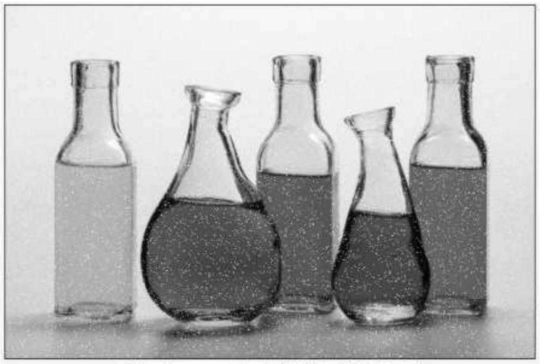
### No arguments

## Description:

Start a `do...while` block.

## Example of use:

```
image.jpg luminance i:=ia+2 do set 255,{u(100)}%,{u(100)}% while
ia<$i
```



[0]: 'image.jpg' (640x427x1x1)

# dog

## Arguments:

- `_sigma1>=0[%],_sigma2>=0[%]`

## Description:

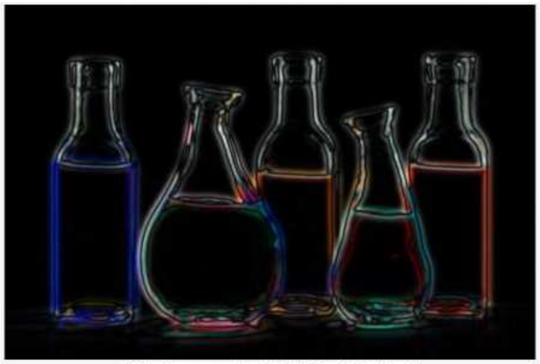
Compute difference of gaussian on selected images.

## Default values:

`sigma1=2%` and `sigma2=3%`.

## Example of use:

```
image.jpg dog 2,3
```



[0]: 'image.jpg' (640x427x1x3)

# done

Built-in command

**No arguments**

## Description:

End a `for/foreach/local/repeat...done` block, and go to associated `for/foreach/repeat` if iterations remain.

(*equivalent to shortcut command* `}`).

---

# double3d

## Arguments:

- `_is_double_sided={ 0 | 1 }`

## Description:

Enable/disable double-sided mode for 3D rendering.

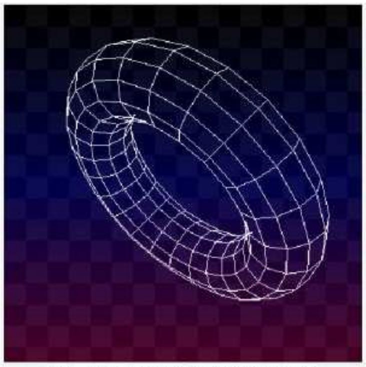
(*equivalent to shortcut command* `db3d`).

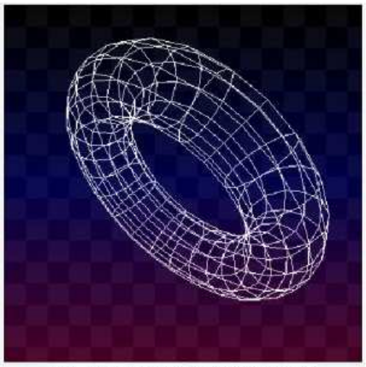
## Default values:

`is_double_sided=1`.

## Example of use:

```
mode3d 1 repeat 2 { torus3d 100,30 rotate3d[-1] 1,1,0,60 double3d $>
snapshot3d[-1] 400 }
```


[0]: '[3D torus]' (400x400x1x3)


[1]: '[3D torus]' (400x400x1x3)

---

# draw_whirl

## Arguments:

- `_amplitude>=0`

## Description:

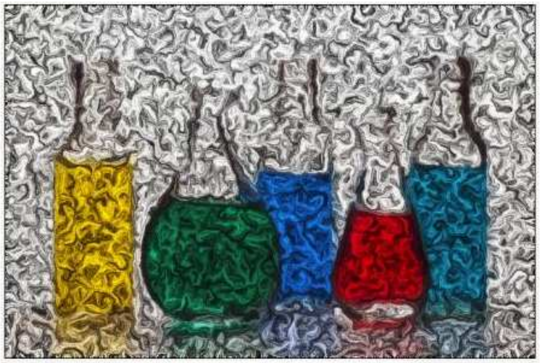
Apply whirl drawing effect on selected images.

### Default values:

`amplitude=100`.

### Example of use:

```
image.jpg draw_whirl ,
```



[0]: 'image.jpg' (640x427x1x3)

## drawing

### Arguments:

- `_amplitude>=0`

### Description:

Apply drawing effect on selected images.

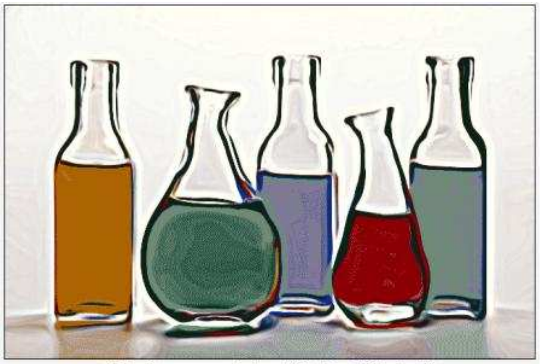
### Default values:

`amplitude=200`.

### Example of use:

```
image.jpg +drawing ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## drop_shadow

### Arguments:

- `_offset_x[%],_offset_y[%],_smoothness[%]>=0,curvature_x>=0,curvature_y>=0,_ex 0 | 1 },_output_separate_layers={ 0 | 1 }`

### Description:

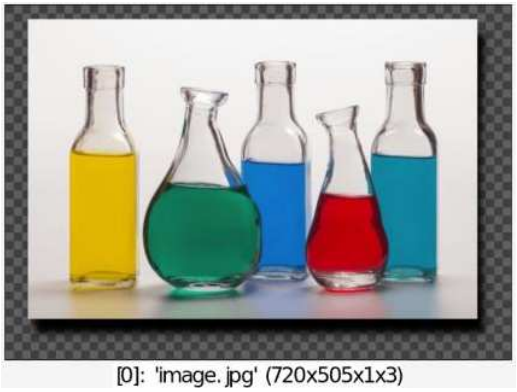
Drop shadow behind selected images.

### Default values:

`offset_x=20`, `offset_y=offset_x`, `smoothness=5`, `curvature_x=curvature_y=0`, `expand_size=1` and `output_separate_layers=0`.

### Example of use:

```
image.jpg drop_shadow 10,20,5,0.5 display_rgba
```



[0]: 'image.jpg' (720x505x1x3)

## drop_shadow

### Arguments:

- `_offset_x[%],_offset_y[%],_smoothness[%]>=0,curvature_x>=0,curvature_y>=0,_ex 0 | 1 },_output_separate_layers={ 0 | 1 }`

### Description:

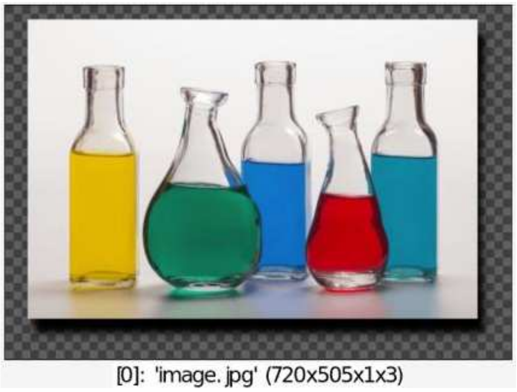
Drop shadow behind selected images.

### Default values:

`offset_x=20`, `offset_y=offset_x`, `smoothness=5`, `curvature_x=curvature_y=0`, `expand_size=1` and `output_separate_layers=0`.

### Example of use:

```
image.jpg drop_shadow 10,20,5,0.5 display_rgba
```



[0]: 'image.jpg' (720x505x1x3)

## echo

Built-in command

### Arguments:

- `message`

### Description:

Output specified message on the error output.

(*equivalent to shortcut command* `e`).

Command selection (if any) stands for displayed call stack subset instead of image indices.
When invoked with a `+` prefix (i.e. `+echo`), the command output its message on stdout rather than stderr.

## echo_file

### Arguments:

- `filename,message`

### Description:

Output specified message, appending it to specified output file.

(similar to `echo` for specified output file stream).

## edgels

### Arguments:

- `x0,y0,_n0,_is_high_connectivity={ 0 | 1 }`

### Description:

Extract one or several lists of edgels (and their normals) that defines a 2D binary silhouette.

When specified (i.e. `!=-1`), arguments `x0,y0,n0` are the coordinates of the starting edgel, which must be located on an edge of the binary silhouette.

- If `x0,y0` and `n0` are specified, only a single list of edgels is returned.
- If only `x0,y0` are specified (meaning `n0=-1`), up to 4 lists of edgels can be returned, all starting from the same point (x0,y0).
- If no arguments are specified (meaning `x0=y0=n0=-1`), all possible lists of edgels are returned.

A list of edgels is returned as an image with 3 channels `[x,y,n]` where `x` and `y` are the 2D coordinates of the edgel pixel, and `n` is the orientation of its associated canonical normal (which can be ***{ 0:[1,0] | 1:[0,1] | 2:[-1,0] | 3:[0,-1] }***.

### Default values:

`x0=y0=n0=-1` and `is_high_connectivity=1`.

---

## edges

### Arguments:

- `_threshold[%]>=0`

### Description:

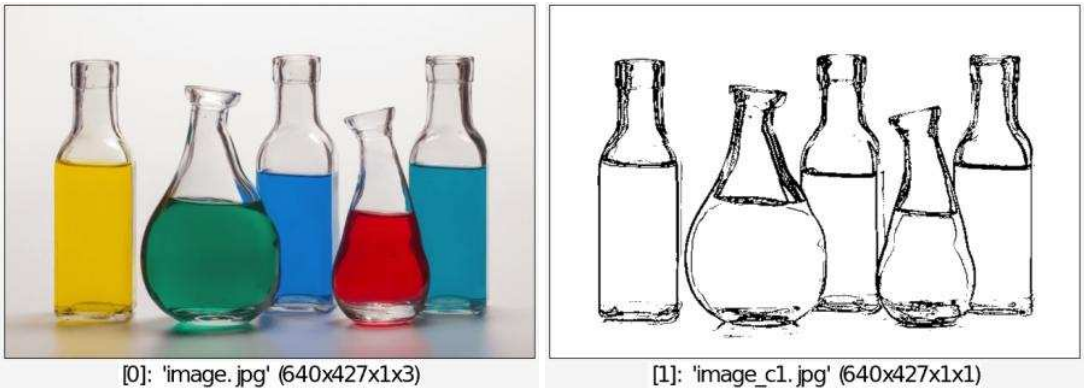
Estimate contours of selected images.

### Default values:

`edges=15%`

### Example of use:

```
image.jpg +edges 15%
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x1)

# eigen

Built-in command

**No arguments**

## Description:

Compute the eigenvalues and eigenvectors of selected symmetric matrices or matrix fields.

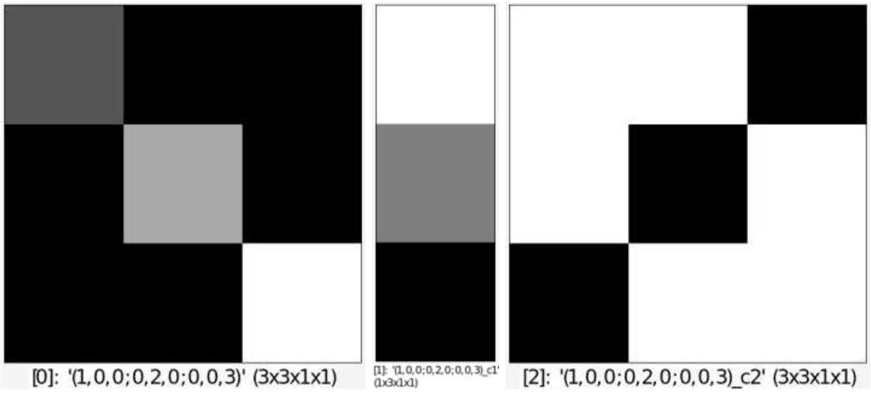
If one selected image has 3 or 6 channels, it is regarded as a field of 2x2 or 3x3 symmetric matrices, whose eigen elements are computed at each point of the field.

This command has a **tutorial page**.

## Examples of use:

### • Example #1

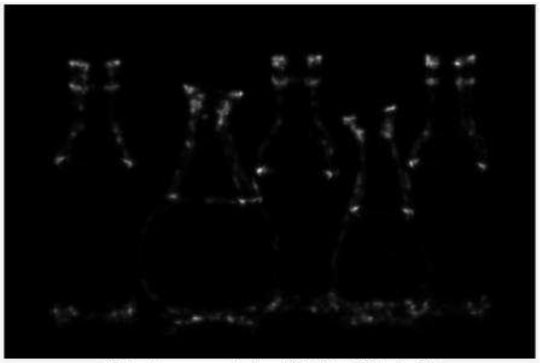
```
(1,0,0;0,2,0;0,0,3) +eigen
```


[0]: '(1,0,0;0,2,0;0,0,3)' (3x3x1x1)
[1]: '(1,0,0;0,2,0;0,0,3)_c1' (1x3x1x1)
[2]: '(1,0,0;0,2,0;0,0,3)_c2' (3x3x1x1)

### • Example #2

```
image.jpg structuretensors blur 2 eigen split[0] c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)

[2]: 'image_c1.jpg' (640x427x1x2)

# eigen2tensor

**No arguments**

## Description:

Recompose selected pairs of eigenvalues/eigenvectors as 2x2 or 3x3 tensor fields.

This command has a **tutorial page**.

# elevate

## Arguments:

- `_depth,_is_plain={ 0 | 1 },_is_colored={ 0 | 1 }`

## Description:

Elevate selected 2D images into 3D volumes.

## Default values:

`depth=64`, `is_plain=1` and `is_colored=1`.

# elevation3d

## Arguments:

- `{ z-factor | [elevation_map] | 'formula' },base_height={ -1 | >=0 }` or
- `(no arg)`

## Description:

Generate 3D elevation of selected images, opt. with a specified elevation map.

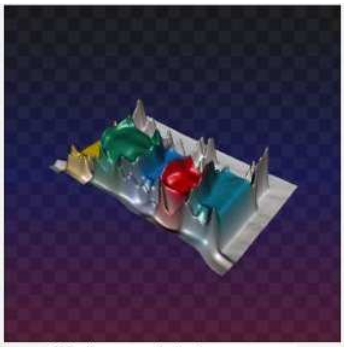
When invoked with (no arg) or `z-factor`, the elevation map is computed as the pointwise L2 norm of the
pixel values. Otherwise, the elevation map is taken from the specified image or formula.

## Examples of use:

### • Example #1

```
image.jpg +blur 5 elevation3d. 0.75
```

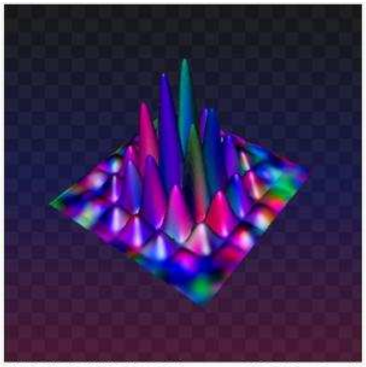
[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg'
(273280 vert., 272214 prim.)

### • Example #2

```
128,128,1,3,u(255) plasma 10,3 blur 4 sharpen 10000 n 0,255
elevation3d[-1] 'X=(x-64)/6;Y=(y-64)/6;-100*exp(-(X^2+Y^2)/
30)*abs(cos(X)*sin(Y))'
```



[0]: '[u(255)]' (16384 vert., 16129 prim.)

# elif

Built-in command

## Arguments:

- `condition`

## Description:

Start a `elif...[else]...fi` block if previous `if` was not verified

and test if specified condition holds
`condition` is a mathematical expression, whose evaluation is interpreted as ***`{ 0:false | other:true }`***..

This command has a **tutorial page**.

# ellipse

Built-in command

## Arguments:

- `x[%],y[%],R[%],r[%],_angle,_opacity,_pattern,_color1,...`

## Description:

Draw specified colored ellipse on selected images.

A radius of `100%` stands for `sqrt(width^2+height^2)`.
`pattern` is an hexadecimal number starting with `0x` which can be omitted
even if a color is specified. If a pattern is specified, the ellipse is
drawn outlined instead of filled.

## Default values:

`opacity=1`, `pattern=(undefined)` and `color1=0`.

## Example of use:

```
image.jpg repeat 300 ellipse {u(100)}%,{u(100)}%,{u(30)},{u(30)},
{u(180)},0.3,${-rgb} done ellipse 50%,50%,100,100,0,0.7,255
```

[0]: 'image.jpg' (640x427x1x3)

---

# ellipsionism

## Arguments:

- `_R>0[%],_r>0[%],_smoothness>=0[%],_opacity,_outline>0,_density>0`

## Description:

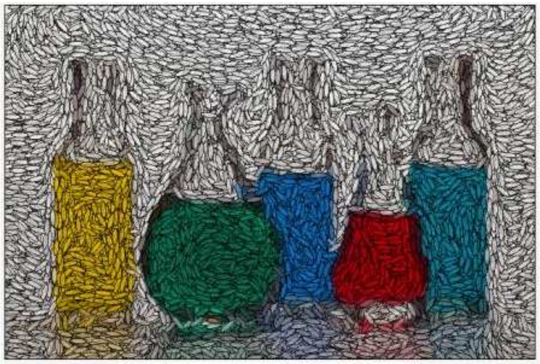
Apply ellipsionism filter to selected images.

## Default values:

`R=10`, `r=3`, `smoothness=1%`, `opacity=0.7`, `outline=8` and `density=0.6`.

## Example of use:

```
image.jpg ellipsionism ,
```



[0]: 'image.jpg' (640x427x1x3)

---

# else

**Built-in command**

### No arguments

## Description:

Execute following commands if previous `if` or `elif` conditions failed.

This command has a **tutorial page**.

## empty3d

**No arguments**

### Description:

Input empty 3D object.

### Example of use:

```
empty3d
```

[0]: '[3D empty]' (0 vert., 0 prim.)

## endian

**Built-in command**

### Arguments:

- `_datatype`

### Description:

Reverse data endianness of selected images, eventually considering the pixel being of the specified datatype.

`datatype` can be *{ bool | uint8 | int8 | uint16 | int16 | uint32 | int32 | uint64 | int64 | float32 | float64 }*.
This command does nothing for `bool`, `uint8` and `int8` datatypes.

## eq

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

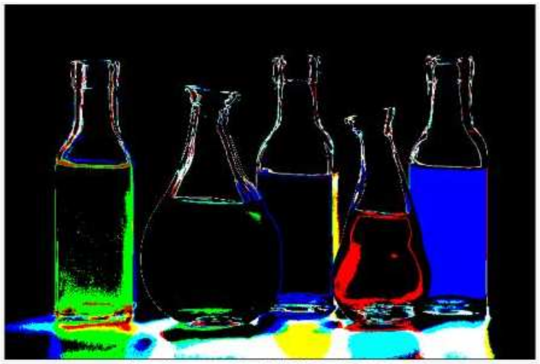
Compute the boolean equality of selected images with specified value, image or mathematical expression, or compute the boolean equality of selected images.

(*equivalent to shortcut command* `==`).

## Examples of use:

### • Example #1

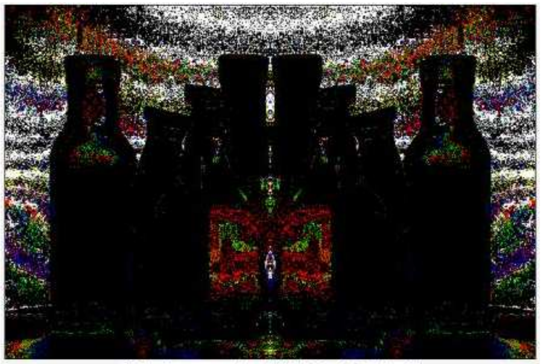
```
image.jpg round 40 eq {round(ia,40)}
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +mirror x eq
```



[0]: 'image.jpg' (640x427x1x3)

# equalize

**Built-in command**

## Arguments:

- `_nb_levels>0[%],_value_min[%],_value_max[%]` or
- `(no arg)`

## Description:

Equalize histograms of selected images.

If value range is specified, the equalization is done only for pixels in the specified value range.

## Default values:

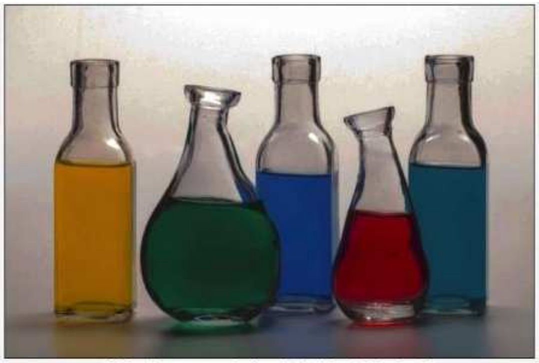
`nb_levels=256`, `value_min=0%` and `value_max=100%`.

## Examples of use:

### • Example #1

```
image.jpg +equalize
```

[0]: 'image.jpg' (640x427x1x3)

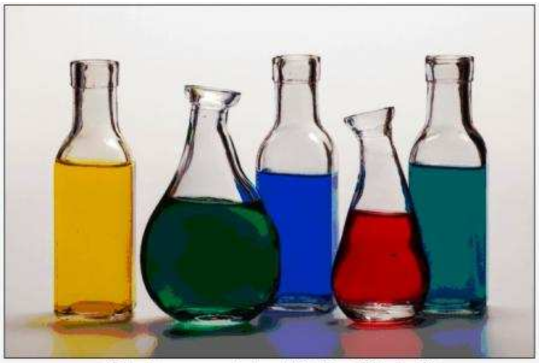


[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +equalize 4,0,128
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# equirectangular2nadirzenith

### No arguments

## Description:

Transform selected equirectangular images to nadir/zenith rectilinear projections.

# erf

Built-in command

### No arguments

## Description:

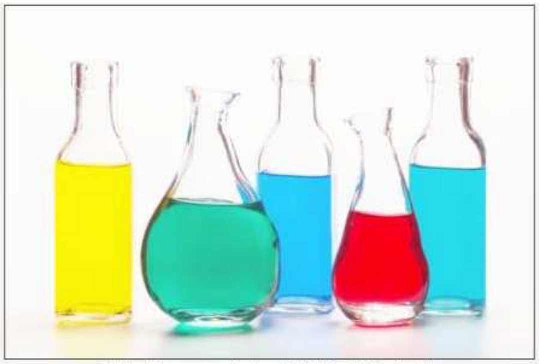
Compute the pointwise error function of selected images.

## Examples of use:

### • Example #1

```
image.jpg +normalize 0,2 erf[-1]
```

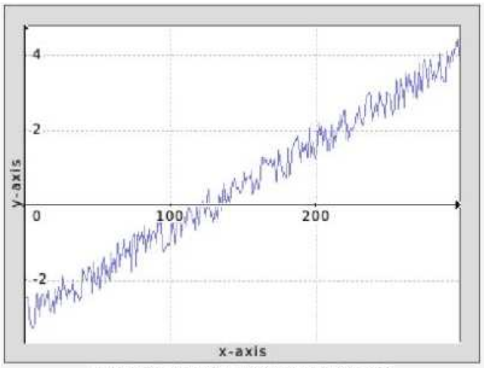
[0]: 'image.jpg' (640x427x1x3)



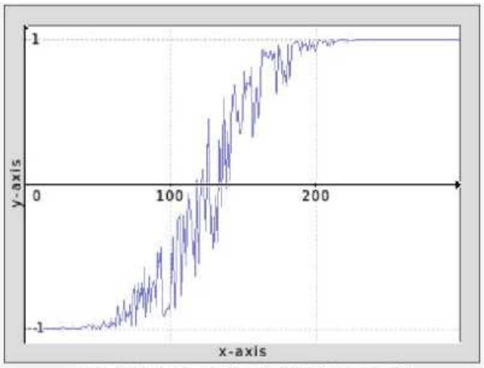
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'7*x/w-3.5+u' +erf display_graph 400,300
```



[0]: '[7*x/w-3.5+u]' (400x300x1x3)



[1]: '[7*x/w-3_c1.5+u]' (400x300x1x3)

# erode

Built-in command

## Arguments:

- `size>=0` or
- `size_x>=0,size_y>=0,_size_z>=0` or
- `[kernel],_boundary_conditions,_is_real={ 0:binary-mode | 1:real-mode }`

## Description:

Erode selected images by a rectangular or the specified structuring element.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

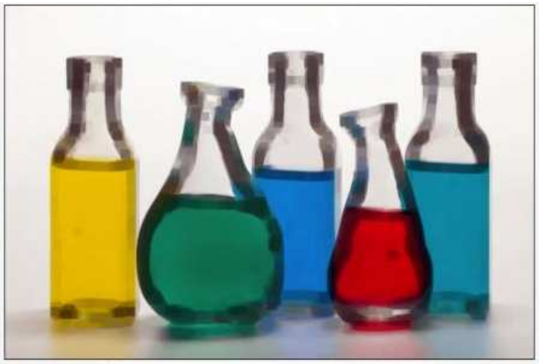
## Default values:

`size_z=1`, `boundary_conditions=1` and `is_real=0`.

## Example of use:

```
image.jpg +erode 10
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# erode_circ

## Arguments:

- `_size>=0,_boundary_conditions,_is_real={ 0 | 1 }`

## Description:

Apply circular erosion of selected images by specified size.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

### Default values:

`boundary_conditions=1` and `is_real=0`.

### Example of use:

```
image.jpg +erode_circ 7
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

## erode_oct

### Arguments:

- `_size>=0,_boundary_conditions,_is_real={ 0 | 1 }`

### Description:

Apply octagonal erosion of selected images by specified size.

### Default values:

`boundary_conditions=1` and `is_real=0`.

### Example of use:

```
image.jpg +erode_oct 7
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

# erode_threshold

## Arguments:

- `size_x>=1,size_y>=1,size_z>=1,_threshold>=0,_boundary_conditions`

## Description:

Erode selected images in the (X,Y,Z,I) space.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

## Default values:

`size_y=size_x`, `size_z=1`, `threshold=255` and `boundary_conditions=1`.

# error

Built-in command

## Arguments:

- `message`

## Description:

Print specified error message on the standard error (stderr) and exit interpreter, except

if error is caught by a `onfail` command.
Command selection (if any) stands for displayed call stack subset instead of image indices.

# euclidean2polar

## Arguments:

- `_center_x[%],_center_y[%],_stretch_factor>0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Apply euclidean to polar transform on selected images.

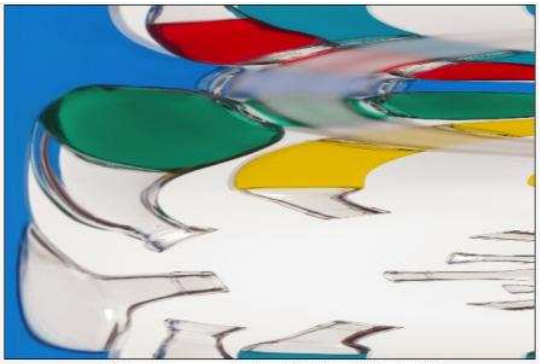
## Default values:

`center_x=center_y=50%`, `stretch_factor=1` and `boundary_conditions=3`.

## Example of use:

```
image.jpg +euclidean2polar ,
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

# eval

Built-in command

## Arguments:

- `expression`

## Description:

Evaluate specified math expression.

- If no command selection is specified, the expression is evaluated once and its result is set to status.
- If command selection is specified, the evaluation is looped over selected images. Status is unchanged. In this case, `eval` is similar to **fill** without assigning the image values.

# exec

Built-in command

## Arguments:

- `_is_verbose={ 0 | 1 },"command"`

## Description:

Execute external command using a system call.

The status value is then set to the error code returned by the system call.
If `is_verbose=1`, the executed command is allowed to output on stdout/stderr.

(*equivalent to shortcut command* `x`).

## Default values:

`is_verbose=1`.

# exec_out

## Arguments:

- `_mode,"command"`

## Description:

Execute external command using a system call, and return resulting `stdout` and/or `stderr`.

`mode` can be ***{ 0:stdout | 1:stderr | 2:stdout+stderr }***.

# exp

**Built-in command**

### No arguments

## Description:

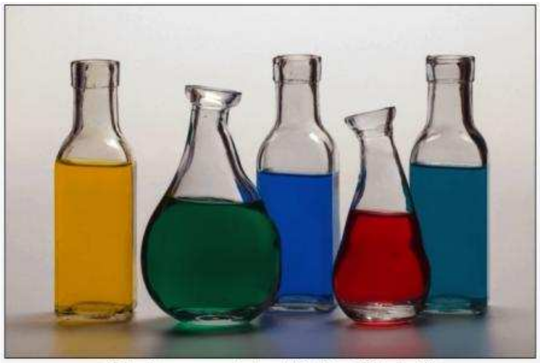
Compute the pointwise exponential of selected images.

## Examples of use:

### • Example #1

```
image.jpg +normalize 0,2 exp[-1]
```

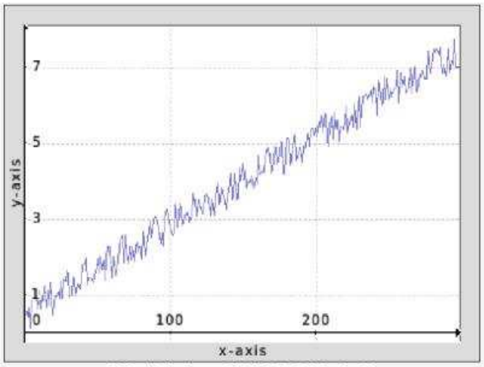
[0]: 'image.jpg' (640x427x1x3)



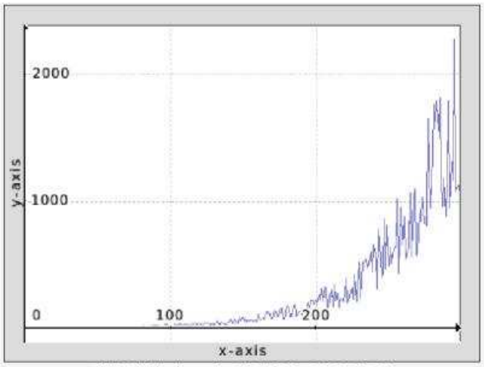
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'7*x/w+u' +exp display_graph 400,300
```



[0]: '[7*x/w+u]' (400x300x1x3)



[1]: '[7*x/w+u]_c1' (400x300x1x3)

# expand_x

## Arguments:

- `size_x>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Expand selected images along the x-axis.

## Default values:

`boundary_conditions=0`.

## Example of use:

```
image.jpg expand_x 30,0
```

[0]: 'image.jpg' (700x427x1x3)

# expand_xy

## Arguments:

- `size>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic |`

`3:mirror }`

### Description:

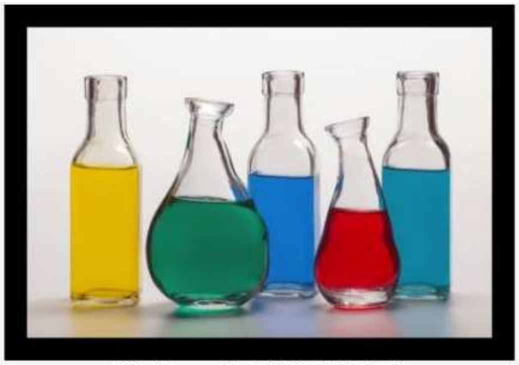
Expand selected images along the xy-axes.

### Default values:

`boundary_conditions=0`.

### Example of use:

```
image.jpg expand_xy 30,0
```



[0]: 'image.jpg' (700x487x1x3)

---

## expand_xyz

### Arguments:

- `size>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

### Description:

Expand selected images along the xyz-axes.

### Default values:

`boundary_conditions=0`.

---

## expand_y

### Arguments:

- `size_y>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

### Description:

Expand selected images along the y-axis.

### Default values:

`boundary_conditions=0`.

### Example of use:

```
image.jpg expand_y 30,0
```

[0]: 'image.jpg' (640x487x1x3)

## expand_z

### Arguments:

- `size_z>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

### Description:

Expand selected images along the z-axis.

### Default values:

`boundary_conditions=0`.

## extract

### Arguments:

- `"condition",_output_type={ 0:xyzc-coordinates | 1:xyz-coordinates | 2:scalar-values | 3:vector-values }`

### Description:

Extract a list of coordinates or values from selected image, where

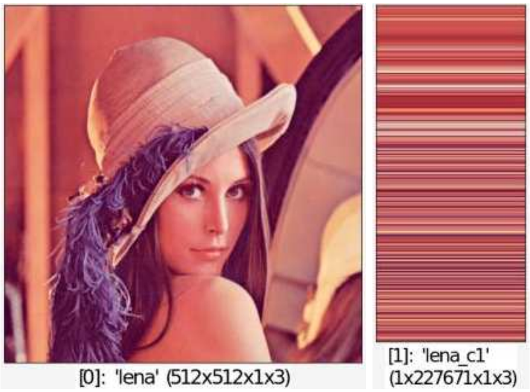
specified mathematical condition holds.
For N coordinates matching, result is a 1xNx1x4 image.

## Default values:

`output_type=0`.

## Example of use:

```
sp lena +extract "norm(I)>128",3
```



---

# extract_region

## Arguments:

- `[label_image],_extract_xyz_coordinates={ 0 | 1 },_label_1,...,_label_M`

## Description:

Extract all pixels of selected images whose corresponding label in `[label_image]` is equal to `label_m`,

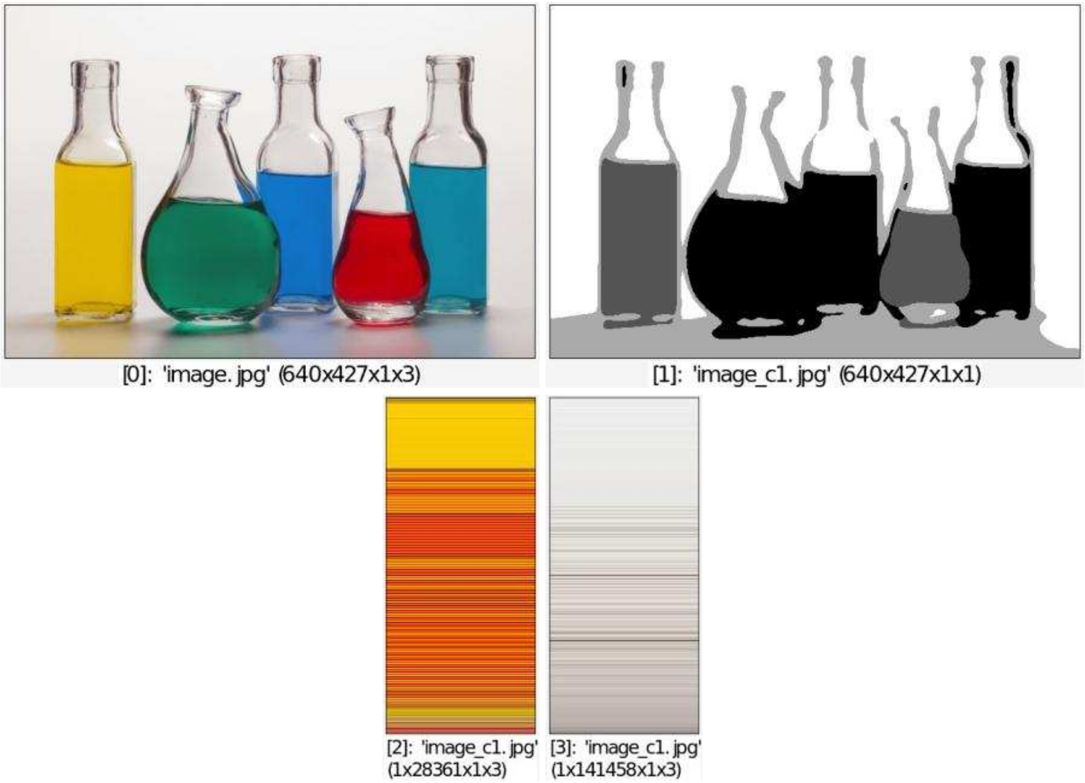
and output them as M column images.

## Default values:

`extract_xyz_coordinates=0`.

## Example of use:

```
image.jpg +blur 3 quantize. 4,0 +extract_region[0] [1],0,1,3
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x1)

[2]: 'image_c1.jpg' (1x28361x1x3)

[3]: 'image_c1.jpg' (1x141458x1x3)

# extract_textures3d

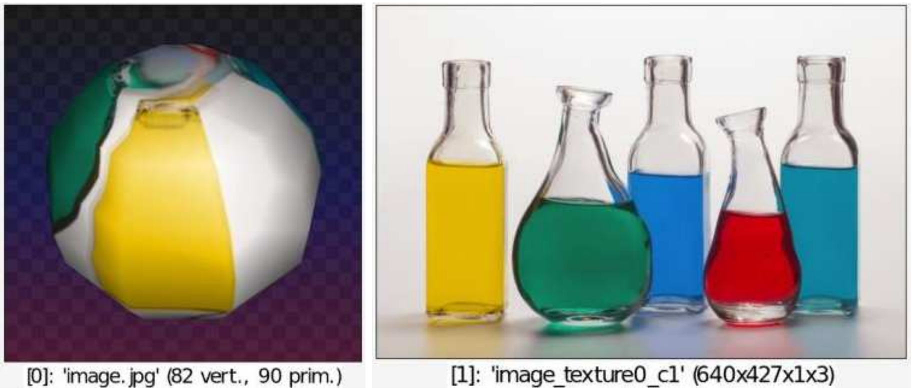
**No arguments**

## Description:

Extract texture data from selected 3D objects.

## Example of use:

```
image.jpg imagesphere3d 10,10 +extract_textures3d
```



[0]: 'image.jpg' (82 vert., 90 prim.)

[1]: 'image_texture0_c1' (640x427x1x3)

# extrude3d

## Arguments:

- `_depth>0,_resolution>0,_smoothness[%]>=0`

## Description:

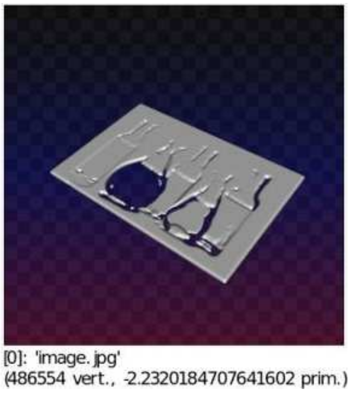
Generate extruded 3D object from selected binary XY-profiles.

## Default values:

`depth=16`, `resolution=1024` and `smoothness=0.5%`.

## Example of use:

```
image.jpg threshold 50% extrude3d 16
```



```
[0]: 'image. jpg'
(486554 vert., -2.2320184707641602 prim.)
```

---

# eye

## Arguments:

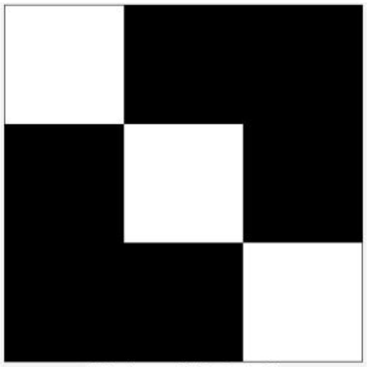
- `_size>0`

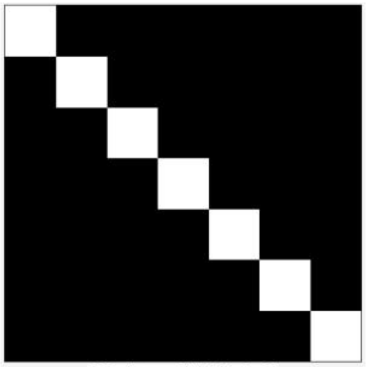
## Description:

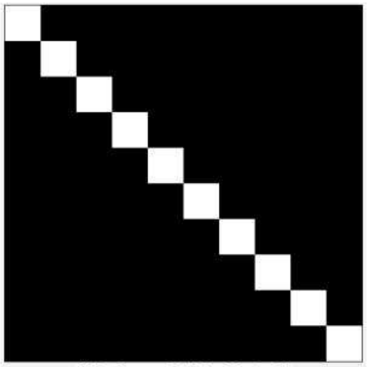
Insert an identity matrix of given size at the end of the image list.

## Example of use:

```
eye 3 eye 7 eye 10
```


[0]: '[x==y]' (3x3x1x1)


[1]: '[x==y]' (7x7x1x1)


[2]: '[x==y]' (10x10x1x1)

# fade_diamond

## Arguments:

- `0<=_start<=100,0<=_end<=100`

## Description:

Create diamond fading from selected images.

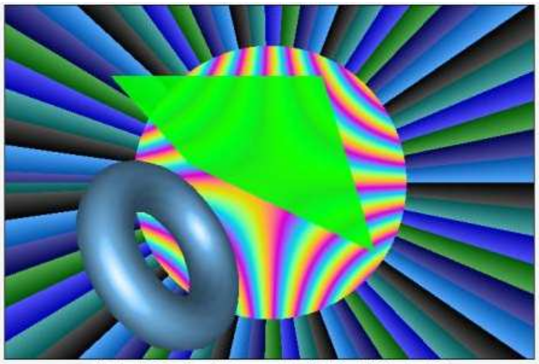
## Default values:

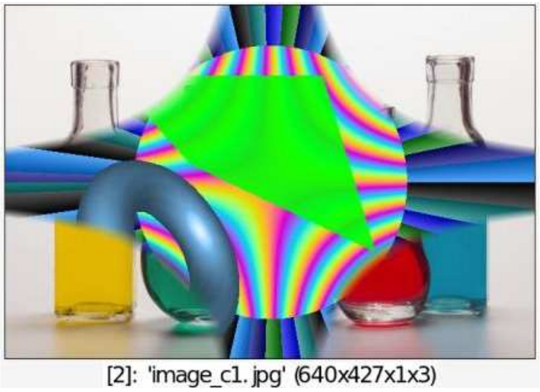
`start=80` and `end=90`.

## Example of use:

```
image.jpg testimage2d {w},{h} +fade_diamond 80,85
```

[0]: 'image.jpg' (640x427x1x3)


[1]: '[2D test image]' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

---

# fade_files

## Arguments:

- `"filename_pattern",_nb_inner_frames>0,_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1,_output_filename`

## Description:

Generate a temporal fading from specified input image files, in a streamed way.

If a display window is opened, rendered frames are displayed in it during processing.
The output filename may have extension `avi` or `mp4` (saved as a video), or any other usual image file extension (saved as a sequence of images).

## Default values:

`nb_inner_frames=10`, `first_frame=0`, `last_frame=-1`, `frame_step=1` and `output_filename=(undefined)`.

---

# fade_linear

## Arguments:

- `_angle,0<=_start<=100,0<=_end<=100`

## Description:

Create linear fading from selected images.

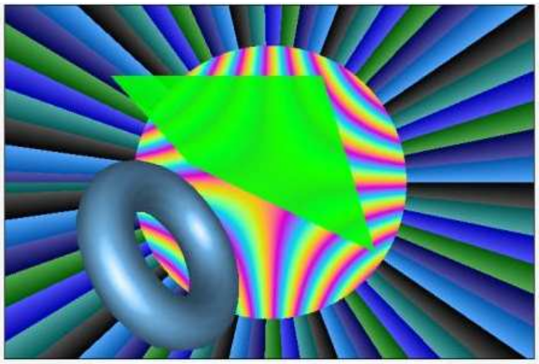
## Default values:

`angle=45`, `start=30` and `end=70`.

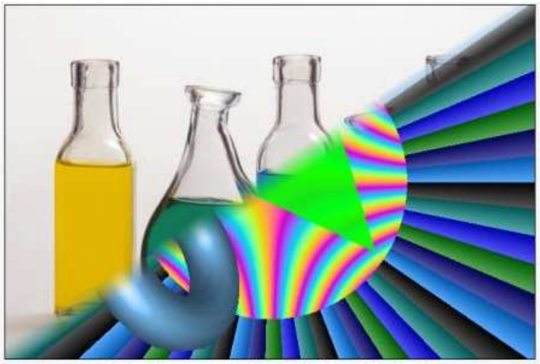
## Example of use:

```
image.jpg testimage2d {w},{h} +fade_linear 45,48,52
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[2D test image]' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

---

# fade_radial

## Arguments:

- `0<=_start<=100,0<=_end<=100`

## Description:

Create radial fading from selected images.

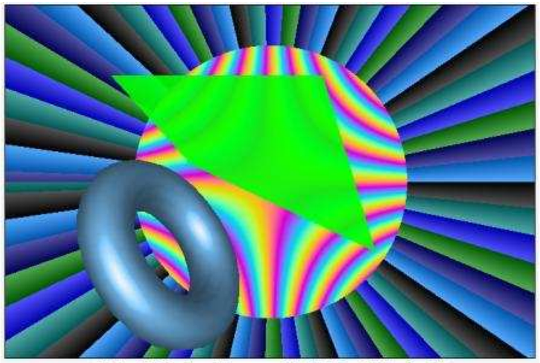
## Default values:

`start=30` and `end=70`.

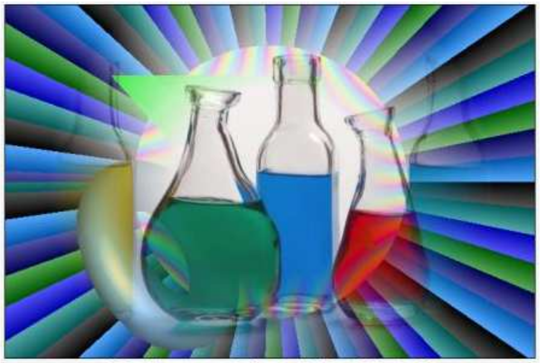
## Example of use:

```
image.jpg testimage2d {w},{h} +fade_radial 30,70
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[2D test image]' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# fade_video

## Arguments:

- `video_filename,_nb_inner_frames>0,_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1,_output_filename`

## Description:

Create a temporal fading sequence from specified input video file, in a streamed way.

If a display window is opened, rendered frames are displayed in it during processing.
This command requires features from the OpenCV library (not enabled in G'MIC by default).

## Default values:

`nb_inner_frames=10`, `first_frame=0`, `last_frame=-1`, `frame_step=1` and `output_filename=(undefined)`.

# fade_x

## Arguments:

- `0<=_start<=100,0<=_end<=100`

## Description:

Create horizontal fading from selected images.

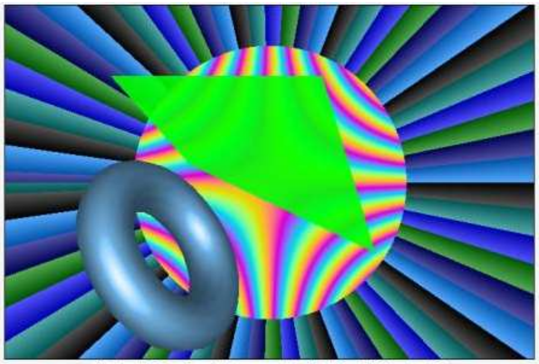
## Default values:

`start=30` and `end=70`.

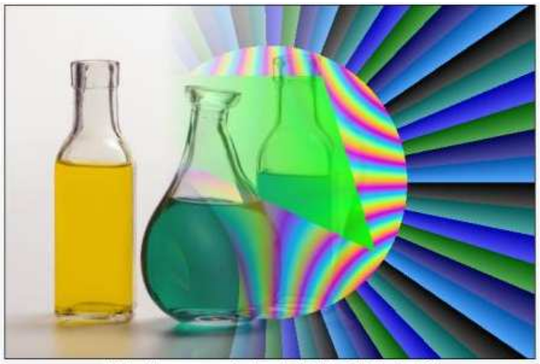
## Example of use:

```
image.jpg testimage2d {w},{h} +fade_x 30,70
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[2D test image]' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

---

# fade_y

## Arguments:

- `0<=_start<=100,0<=_end<=100`

## Description:

Create vertical fading from selected images.

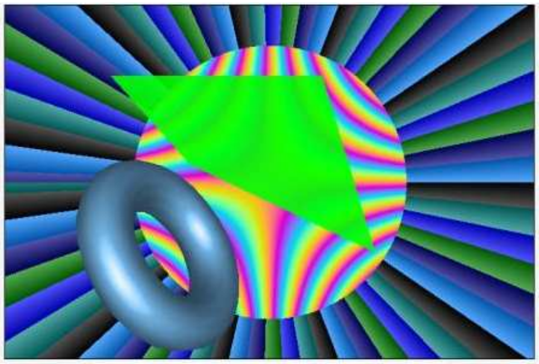
## Default values:

`start=30` and `end=70`.

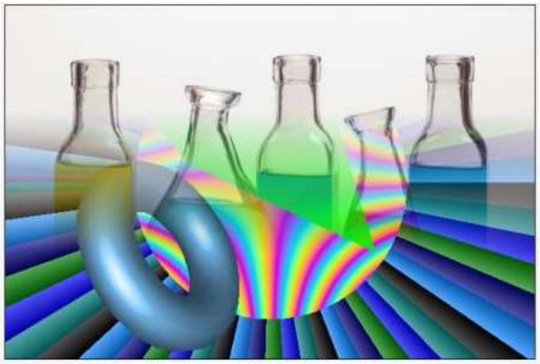
## Example of use:

```
image.jpg testimage2d {w},{h} +fade_y 30,70
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[2D test image]' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# fade_z

## Arguments:

- `0<=_start<=100,0<=_end<=100`

## Description:

Create transversal fading from selected images.

## Default values:

`start=30` and `end=70`.

# fft

Built-in command

## Arguments:

- `_{ x | y | z }...{ x | y | z }`

## Description:

Compute the direct fourier transform (real and imaginary parts) of selected images,

optionally along the specified axes only.

## See also:

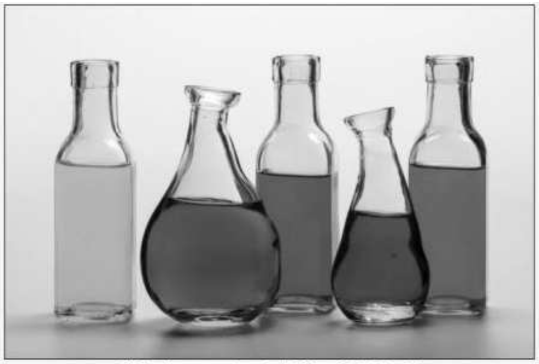
**ifft** .

This command has a **tutorial page**.

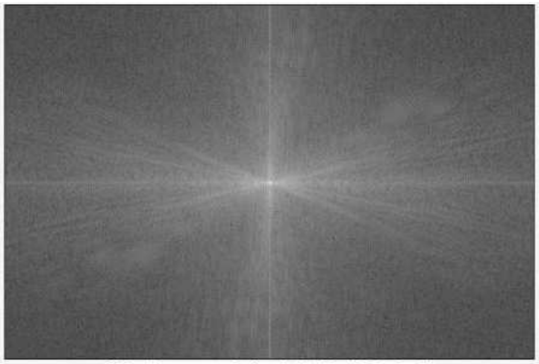
## Examples of use:

### • Example #1

```
image.jpg luminance +fft append[-2,-1] c norm[-1] log[-1] shift[-1]
50%,50%,0,0,2
```



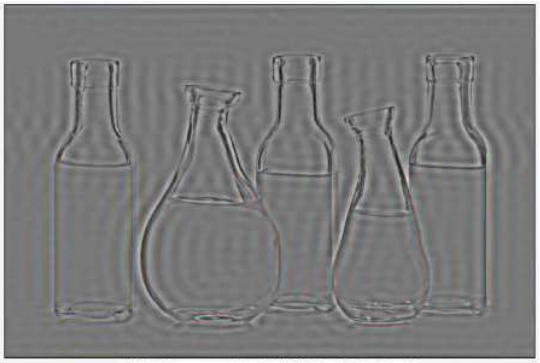
[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)

### • Example #2

```
image.jpg w2:=int(w/2) h2:=int(h/2) fft shift $w2,$h2,0,0,2 ellipse
$w2,$h2,30,30,0,1,0 shift -$w2,-$h2,0,0,2 ifft remove[-1]
```



[0]: 'image.jpg' (640x427x1x3)

---

# fftpolar

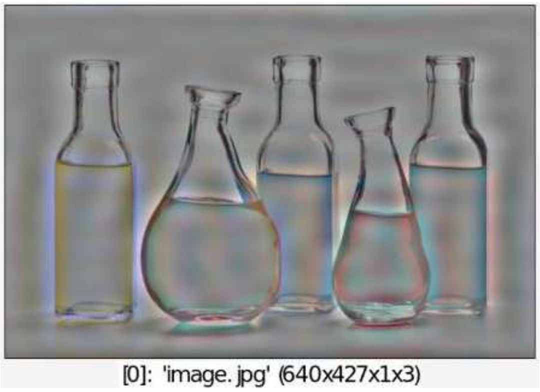
**No arguments**

## Description:

Compute fourier transform of selected images, as centered magnitude/phase images.

## Example of use:

```
image.jpg fftpolar ellipse 50%,50%,10,10,0,1,0 ifftpolar
```



---

# fi

Built-in command

**No arguments**

## Description:

End a `if...[elif]...[else]...fi` block.

(*equivalent to shortcut command* `fi`).

This command has a **tutorial page**.

---

# fibonacci

## Arguments:

- `N>=0`

## Description:

Return the Nth number of the Fibonacci sequence.

## Example of use:

```
echo ${"fibonacci 10"}
```

```
[gmic]-0./ Start G'MIC interpreter.
[gmic]-0./ 55
[gmic]-0./ End G'MIC interpreter.
```

# file_mv

## Arguments:

- `filename_src,filename_dest`

## Description:

Rename or move a file from a location $1 to another location $2.

# filename

## Arguments:

- `filename,_number1,_number2,...,_numberN`

## Description:

Return a filename numbered with specified indices.

# filename_dated

## Arguments:

- `filename`

## Description:

Convert specified filename to one stamped with the current date (`filename_YYYYMMDD_HHMMSS.ext`).

# filename_rand

**No arguments**

## Description:

Return a random filename for storing temporary data.

## files

Built-in command

### Arguments:

- `_mode,path`

### Description:

Return the list of files and/or subfolders from specified path.

`path` can be eventually a matching pattern.
`mode` can be ***{ 0:files only | 1:folders only | 2:files + folders }***.
Add `3` to `mode` to return full paths instead of filenames only.

### Default values:

`mode=5`.

## files2img

### Arguments:

- `_mode,path`

### Description:

Insert a new image where each vector-valued pixel is a string encoding the filenames returned by command **files**.

Useful to manage list of filenames containing characters that have a special meaning in the G'MIC language,such as spaces or commas.

## files2video

### Arguments:

- `"filename_pattern",_output_filename,_fps>0,_codec`

### Description:

Convert several files into a single video file.

### Default values:

`output_filename=output.mp4`, `fps=25` and `codec=mp4v`.

---

# fill

Built-in command

## Arguments:

- `value1,_value2,...` or
- `[image]` or
- `'formula'`

## Description:

Fill selected images with values read from the specified value list, existing image

or mathematical expression. Single quotes may be omitted in `formula`.

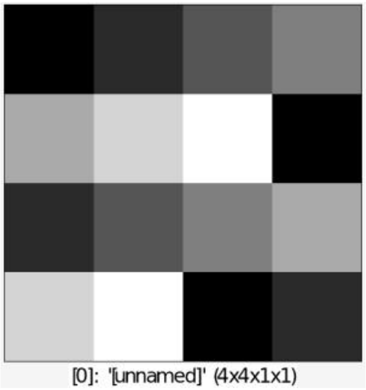
(*equivalent to shortcut command* `f`).

This command has a **tutorial page**.

## Examples of use:

### • Example #1

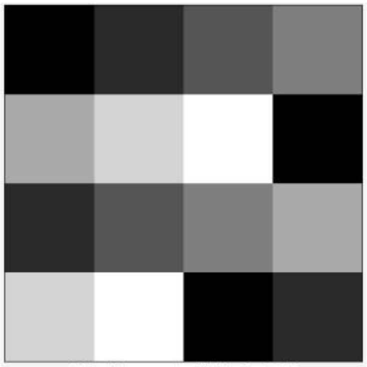
```
4,4 fill 1,2,3,4,5,6,7
```



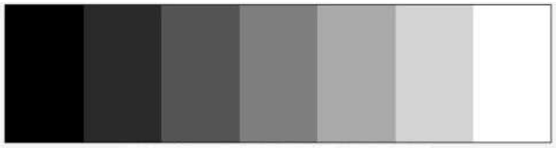
[0]: '[unnamed]' (4x4x1x1)

### • Example #2

```
4,4 (1,2,3,4,5,6,7) fill[-2] [-1]
```



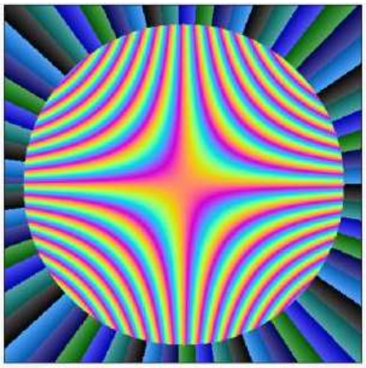
[0]: '[unnamed]' (4x4x1x1)



[1]: '(1,2,3,4,5,6,7)' (7x1x1x1)

- **Example #3**

```
400,400,1,3 fill "X=x-w/2; Y=y-h/2; R=sqrt(X^2+Y^2); a=atan2(Y,X);
R<=180?255*abs(cos(c+200*(x/w-0.5)*(y/h-0.5))):850*(a%(0.1*(c+1)))"
```



[0]: '[unnamed]' (400x400x1x3)

# fill_color

## Arguments:

- `col1,...,colN`

## Description:

Fill selected images with specified color.

(*equivalent to shortcut command* `fc`).

This command has a **tutorial page**.

## Example of use:

```
image.jpg +fill_color 255,0,255
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

---

# fire_edges

## Arguments:

- `_edges>=0,0<=_attenuation<=1,_smoothness>=0,_threshold>=0,_nb_frames>0,_start`

## Description:

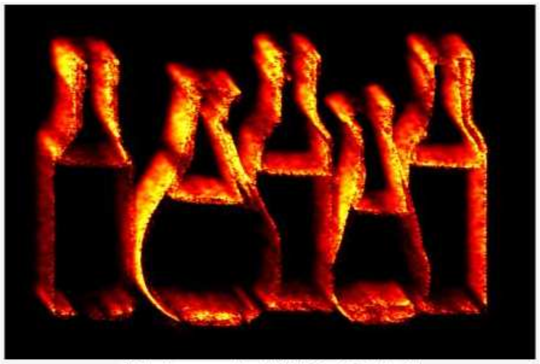
Generate fire effect from edges of selected images.

## Default values:

`edges=0.7`, `attenuation=0.25`, `smoothness=0.5`, `threshold=25`, `nb_frames=1`, `starting_frame=20` and `frame_skip=0`.

## Example of use:

```
image.jpg fire_edges ,
```


[0]: 'image.jpg' (640x427x1x3)

---

# fisheye

## Arguments:

- `_center_x,_center_y,0<=_radius<=100,_amplitude>=0`

## Description:

Apply fish-eye deformation on selected images.

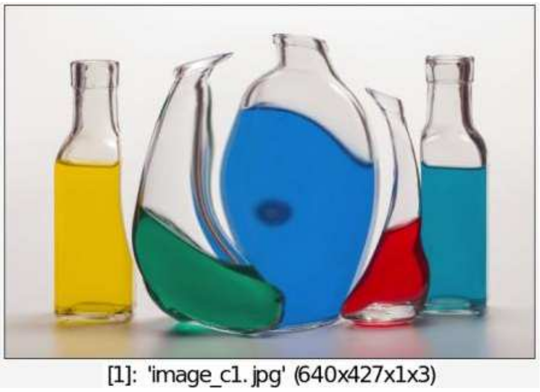
## Default values:

`x=y=50`, `radius=50` and `amplitude=1.2`.

## Example of use:

```
image.jpg +fisheye ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# fitratio_wh

## Arguments:

- `min_width,min_height,ratio_wh`

## Description:

Return a 2D size `width,height` which is bigger than `min_width,min_height` and has the specified w/h ratio.

---

# fitsamples

## Arguments:

- `nb_samples>0,_relevant_dimension[%]>0,_average_vector_varname,_dilation_vecto`

## Description:

Generate `nb_samples` vectors having the same multivariate gaussian distribution as the vectors of the selected images.

Each input represents a set of `M` vectors of dimension `N` (with M>1) (specified as an image with size `MxNx1x1`, `Mx1xNx1`, `Mx1x1xN`, `1xMxNx1`, `1xMx1xN` or `1x1xMxN`).

The command returns a new set of random vectors with similar geometry.

## Default values:

`relevant_dimension=100%`, and `average_vector_varname=orientation_matrix_varname=dilation_matrix_varname=(undefined)`.

# fitscreen

## Arguments:

- `width,height,_depth,_minimal_size[%],_maximal_size[%]` or
- `[image],_minimal_size[%],_maximal_size[%]`

## Description:

Return the `ideal` size WxH for a window intended to display an image of specified size on screen.

## Default values:

`depth=1`, `minimal_size=128` and `maximal_size=85%`.

# flood

**Built-in command**

## Arguments:

- `x[%],_y[%],_z[%],_tolerance>=0,_is_high_connectivity={ 0 | 1 },_opacity,_color1,...`

## Description:

Flood-fill selected images using specified value and tolerance.

## Default values:

`y=z=0`, `tolerance=0`, `is_high_connectivity=0`, `opacity=1` and `color1=0`.

## Example of use:

```
image.jpg repeat 1000 flood {u(100)}%,{u(100)}%,0,20,0,1,${-rgb} done
```

[0]: 'image.jpg' (640x427x1x3)

# flower

## Arguments:

- `_amplitude,_frequency,_offset_r[%],_angle,_center_x[%],_center_y[%],_boundary 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Apply flower deformation on selected images.

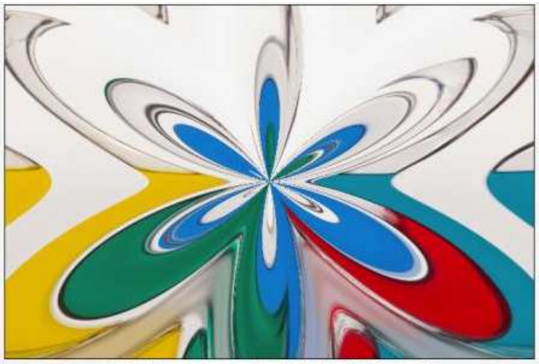
## Default values:

`amplitude=30`, `frequency=6`, `offset_r=0`, `angle=0`, `center_x=center_y=50%` and `boundary_conditions=3`.

## Example of use:

```
image.jpg +flower ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# focale3d

## Arguments:

- `focale`

## Description:

Set 3D focale.

(*equivalent to shortcut command* `f3d`).

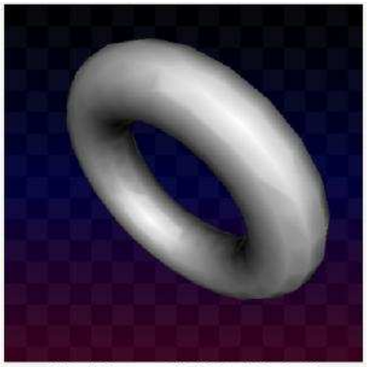
Set `focale` to 0 to enable parallel projection (instead of perspective).
Set negative `focale` will disable 3D sprite zooming.

## Default values:

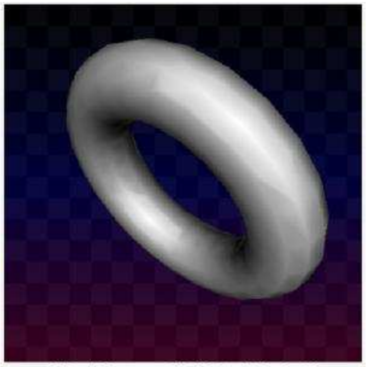
`focale=700`.

## Example of use:

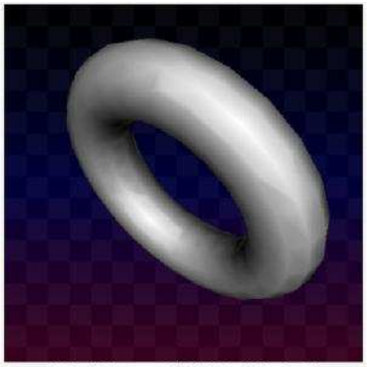
```
repeat 5 { torus3d 100,30 rotate3d[-1] 1,1,0,60 focale3d {$<*90}
snapshot3d[-1] 400 } remove[0]
```



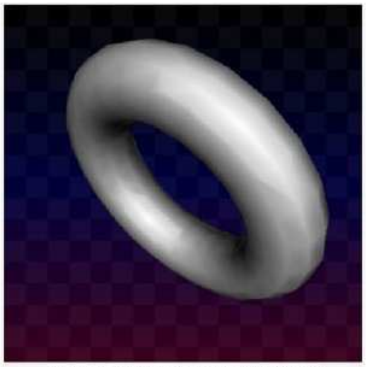
[0]: '[3D torus]' (400x400x1x3)



[1]: '[3D torus]' (400x400x1x3)



[2]: '[3D torus]' (400x400x1x3)



[3]: '[3D torus]' (400x400x1x3)

# font

## Arguments:

- `{ 'Font_name' | font_number | font.gmz },_font_height[%]>0,_is_bold={ 0 | 1 }`

## Description:

Return font identifier (variable name) that can be further used in command `text` as a custom font.

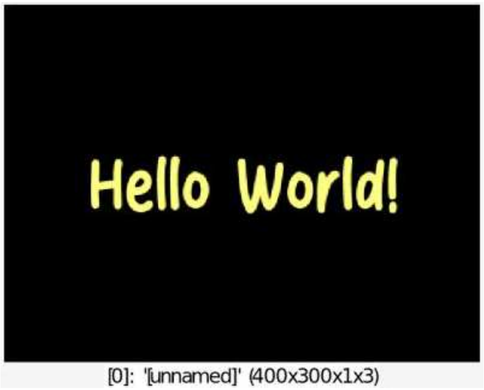
`Font_name` can be *{ Acme | Arial | Arial Black | Black Ops One | BlackChancery | Cabin Sketch | Caprasimo | Carnevalee Freakshow | Cheese Burger | Cheque | Cheque-Black | Chlorinar | Comic Sans MS | Courier New | Creepster | Georgia | Impact | Lobster | Luckiest Guy | Macondo | MedievalSharp | Odin Rounded | Oswald | Palatino Linotype | Playfair Display | Roboto | Sacramento | Satisfy | Sofia | Tex Gyre Adventor | Times New Roman | Titan One | Verdana }*.
If a filename `font.gmz` is specified, it must be a file converted with command `font2gmz`.

## Default values:

`font_height=64` and `is_bold=0`.

## Example of use:

```
400,300,1,3 text "Hello World!",0.5~,0.5~,${"font \"Cheese Burger\",
80"},1,255,255,128
```



[0]: '[unnamed]' (400x300x1x3)

# font2gmz

## Arguments:

- `_font_name,_font_size>0,_font_qualifier`

## Description:

Convert specified font to G'MIC format, so that it can be used as a custom font for command `text`.

`font_name` can be either a filename as `font.ttf`, or a 'Google Font Name'.
This command requires the command line tool `cutycapt` to be installed on your system.
Beware, `font_size` is the size of font used for the rendering, it does **not** correspond to the font height.

## Default values:

`font_name=Sofia`, `font_size=24` and `font_qualifier=""`.

# fontchart

## Arguments:

- `display_mode.`

## Description:

Insert G'MIC font chart at the end of the image list.

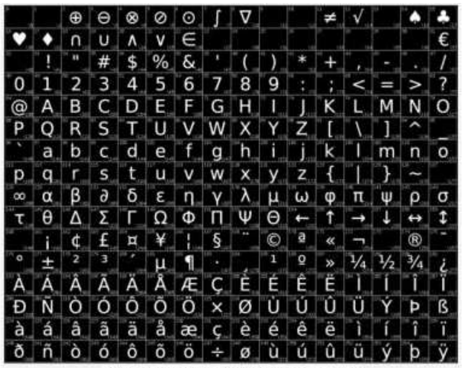
`display_mode` can be ***{ 0: List of characters | N: List of fonts with height 'N'}***.

## Default values:

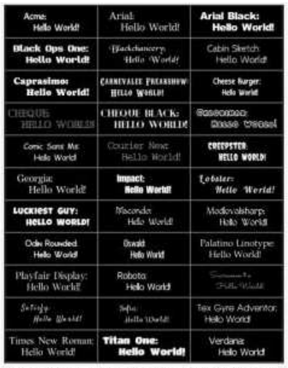
`display_mode=0`.

## Example of use:

```
fontchart 0 fontchart 64
```


⊕ ⊖ ⊗ ⊘ ⊙ ∫ ∇ ≠ √ ♠ ♣
♥ ♦ ∩ ∪ ∧ ∨ ∈ €
! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~
∞ α β ∂ δ ε η γ λ μ ω φ π ψ ρ σ
τ θ Δ Σ Γ Ω Φ Π Ψ Θ ← ↑ → ↓ ↔ ↕
¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ ® ¯
° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
à á â ã ä å æ ç è é ê ë ì í î ï
ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

[0]: '[empty]' (1056x832x1x1)


Acme: Hello World!
Arial: Hello World!
Arial Black: Hello World!
Black Ops One: Hello World!
Blackchancery: Hello World!
Cabin Sketch: Hello World!
Caprasimo: Hello World!
Carnevalee Freakshow: Hello World!
Cheese burger: Hello World
Cheque: Hello World!
Cheque Black: Hello World!
Comic Sans MS: Hello World
Courier New: Hello World!
Creepster: Hello World!
Georgia: Hello World!
Impact: Hello World!
Lobster: Hello World!
Lucky Guy: Hello World!
Macondo: Hello World!
Medievalsharp: Hello World!
Odin Rounded: Hello World
Oswald: Hello World
Palatino Linotype: Hello World!
Playfair Display: Hello World!
Roboto: Hello World
Satisfy: Hello World!
Sofia: Hello World!
Tex Gyre Adventor: Hello World
Times New Roman: Hello World!
Titan One: Hello World!
Verdana: Hello World

[1]: '[empty]' (1581x2024x1x1)

# for

Built-in command

## Arguments:

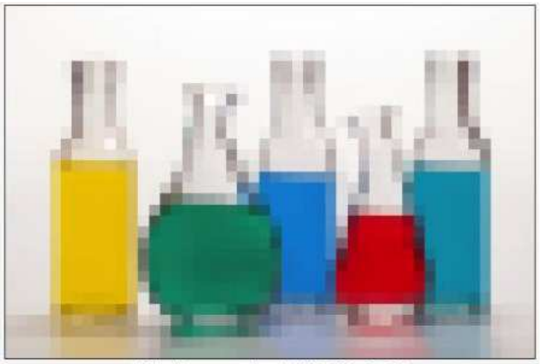
- `condition`

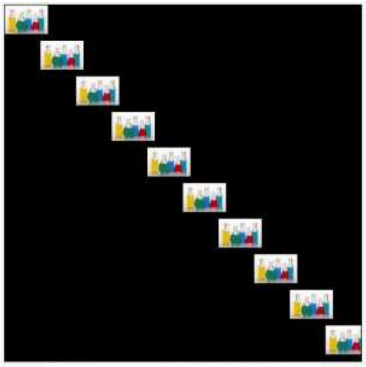
## Description:

Start a `for...done` block.

## Example of use:

```
image.jpg rescale2d ,32 400,400,1,3 x=0 for $x<400 image[1] [0],$x,$x
x+=40 done
```


[0]: 'image.jpg' (48x32x1x3)


[1]: '[unnamed]' (400x400x1x3)

# foreach

Built-in command

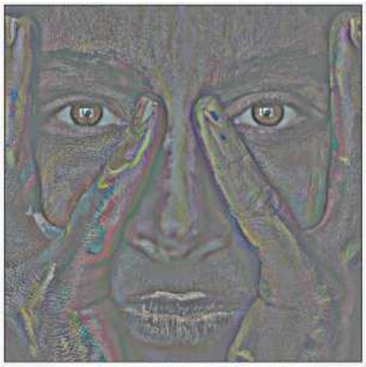
**No arguments**

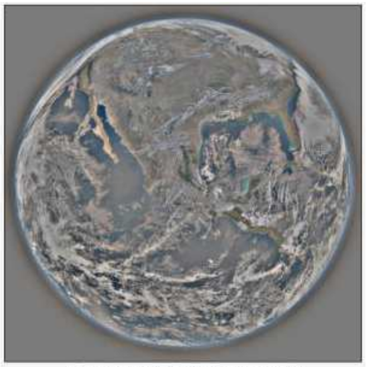
## Description:

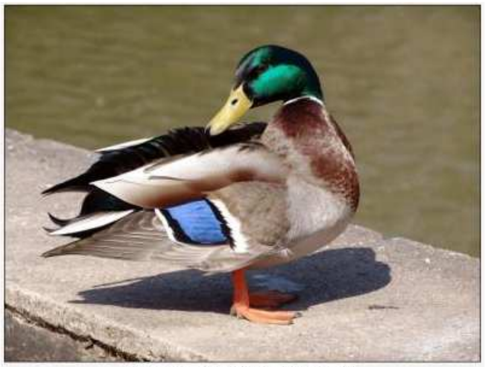
Start a `foreach...done` block, that iterates over all images in the selection, with a separate local environment for each one.

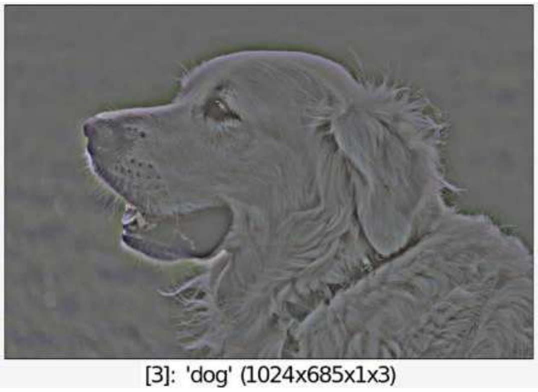
## Example of use:

```
sample colorful,earth,duck,dog foreach[^2] +blur 10 sub normalize
0,255 done
```


[0]: 'colorful' (800x800x1x3)


[1]: 'earth' (500x500x1x3)


[2]: 'duck' (640x480x1x3)



[3]: 'dog' (1024x685x1x3)

# fov3d

## Arguments:

- `fov_angle>=0,_image_resolution>0`

## Description:

Set 3D focale to match specified field of vision angle (in degree) for rendering a 3D object in an image with specified resolution.

Return corresponding value of the focale in status.

## Default values:

`fov_angle=45` and `image_size=max(w,h)` (max size of the latest image).

# fps

**No arguments**

## Description:

Return the number of time this function is called per second, or -1 if this info is not yet available.

Useful to display the framerate when displaying animations.

# fractalize

## Arguments:

- `0<=detail_level<=1`

## Description:

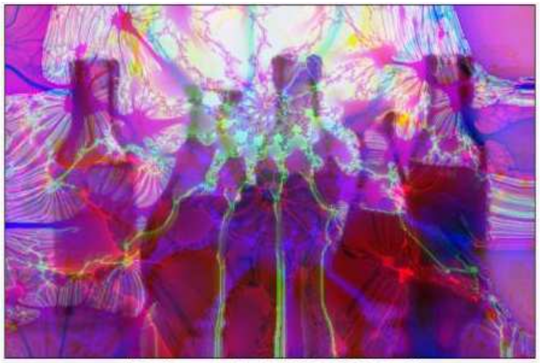
Randomly fractalize selected images.

## Default values:

`detail_level=0.8`

## Example of use:

```
image.jpg fractalize ,
```



[0]: 'image.jpg' (640x427x1x3)

# frame_blur

## Arguments:

- `_sharpness>0,_size>=0,_smoothness,_shading,_blur`

## Description:

Draw RGBA-colored round frame in selected images.

## Default values:

`sharpness=10`, `size=30`, `smoothness=0`, `shading=1` and `blur=3%`.

## Example of use:

```
image.jpg frame_blur 3,30,8,10%
```

[0]: 'image.jpg' (640x427x1x4)

## frame_cube

### Arguments:

- `_depth>=0,_centering_x,_centering_y,_left_side={ 0:normal | 1:mirror-x | 2:mirror-y | 3:mirror-xy },_right_side,_lower_side,_upper_side`

### Description:

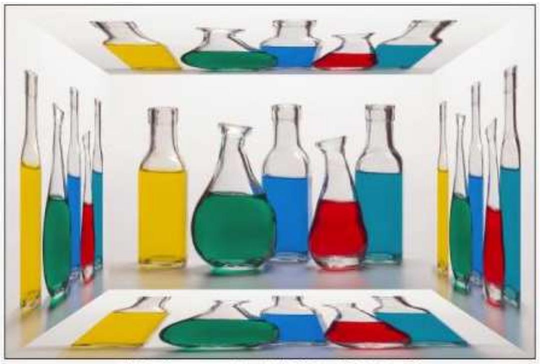
Insert 3D frames in selected images.

### Default values:

`depth=1`, `centering_x=centering_y=0` and `left_side=right_side,lower_side=upper_side=0`.

### Example of use:

```
image.jpg frame_cube ,
```



[0]: 'image.jpg' (640x427x1x3)

## frame_fuzzy

### Arguments:

- `size_x[%]>=0,_size_y[%]>=0,_fuzzyness>=0,_smoothness[%]>=0,_R,_G,_B,_A`

## Description:

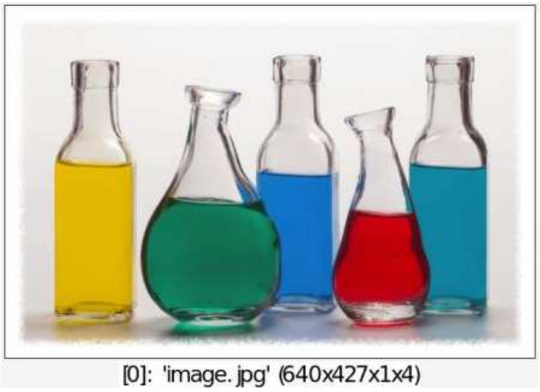
Draw RGBA-colored fuzzy frame in selected images.

## Default values:

`size_y=size_x`, `fuzzyness=5`, `smoothness=1` and `R=G=B=A=255`.

## Example of use:

```
image.jpg frame_fuzzy 20
```



[0]: 'image.jpg' (640x427x1x4)

# frame_painting

## Arguments:

- `_size[%]>=0,0<=_contrast<=1,_profile_smoothness[%]>=0,_R,_G,_B,_vignette_size`

## Description:

Add a painting frame to selected images.

## Default values:

`size=10%`, `contrast=0.4`, `profile_smoothness=6%`, `R=225`, `G=200`, `B=120`, `vignette_size=2%`, `vignette_contrast=400`, `defects_contrast=50`, `defects_density=10`, `defects_size=1`, `defects_smoothness=0.5%` and `serial_number=123456789`.

## Example of use:

```
image.jpg frame_painting ,
```

[0]: 'image.jpg' (768x555x1x3)

---

# frame_pattern

## Arguments:

- `M>=3,_constrain_size={ 0 | 1 }` or
- `M>=3,_[frame_image],_constrain_size={ 0 | 1 }`

## Description:

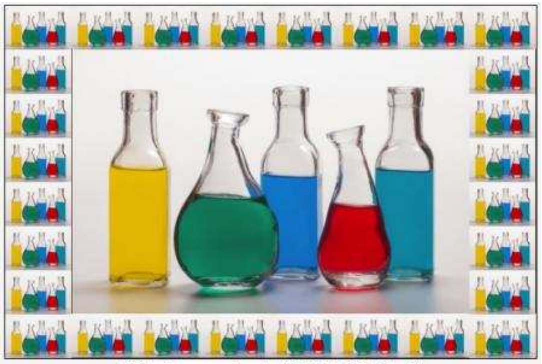
Insert selected pattern frame in selected images.

## Default values:

`pattern=0` and `constrain_size=0`.

## Example of use:

```
image.jpg frame_pattern 8
```



[0]: 'image.jpg' (856x568x1x3)

---

# frame_round

## Arguments:

- `size_x[%]>=0,size_y[%]>=0,radius[%]>=0,_smoothness[%]>=0,_col1,...,_colN`

## Description:

Insert an inner round frame in selected images.

## Default values:

'size_x=size_y=5%, `radius=30%`, `smoothness=0` and `col=0,0,0,255`.

# frame_seamless

## Arguments:

- `frame_size>=0,_patch_size>0,_blend_size>=0,_frame_direction={ 0:inner (preserve image size) | 1:outer }`

## Description:

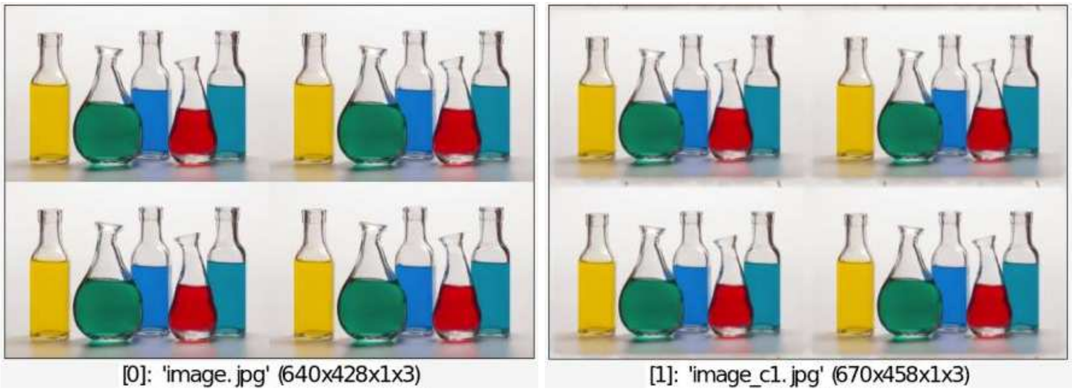
Insert frame in selected images, so that tiling the resulting image makes less visible seams.

## Default values:

`patch_size=7`, `blend_size=5` and `frame_direction=1`.

## Example of use:

```
image.jpg +frame_seamless 30 array 2,2
```


[0]: 'image.jpg' (640x428x1x3)
[1]: 'image_c1.jpg' (670x458x1x3)

# frame_x

## Arguments:

- `size_x[%],_col1,...,_colN`

## Description:

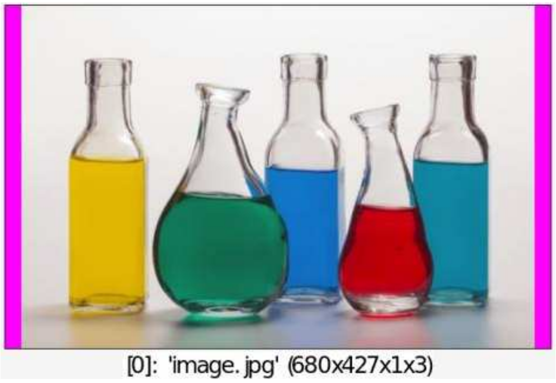
Insert outer frame along the x-axis in selected images.

## Default values:

`col1=col2=col3=255` and `col4=255`.

## Example of use:

```
image.jpg frame_x 20,255,0,255
```



# frame_xy

## Arguments:

- `size_x[%],_size_y[%],_col1,...,_colN`

## Description:

Insert outer frame along the x-axis in selected images.

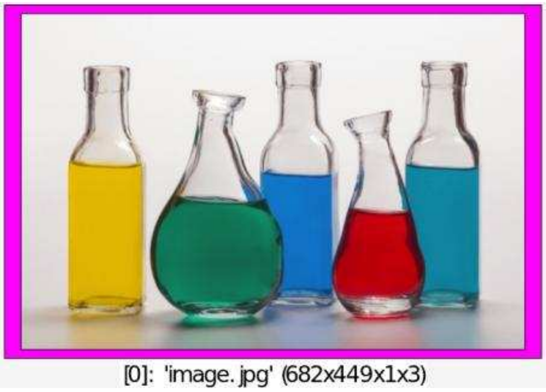
## Default values:

`size_y=size_x`, `col1=col2=col3=255` and `col4=255`.

(*equivalent to shortcut command* `frame`).

## Example of use:

```
image.jpg frame_xy 1,1,0 frame_xy 20,10,255,0,255
```



[0]: 'image.jpg' (682x449x1x3)

---

# frame_xyz

## Arguments:

- `size_x[%],_size_y[%],_size_z[%]_col1,...,_colN`

## Description:

Insert outer frame along the x-axis in selected images.

## Default values:

`size_y=size_x=size_z`, `col1=col2=col3=255` and `col4=255`.

---

# frame_y

## Arguments:

- `size_y[%],_col1,...,_colN`

## Description:

Insert outer frame along the y-axis in selected images.

## Default values:

`col1=col2=col3=255` and `col4=255`.

## Example of use:

```
image.jpg frame_y 20,255,0,255
```

[0]: 'image.jpg' (640x467x1x3)

# function1d

## Arguments:

- `0<=smoothness<=1,x0>=0,y0,x1>=0,y1,...,xn>=0,yn`

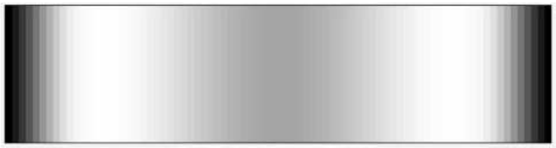
## Description:

Insert continuous 1D function from specified list of keypoints (xk,yk)

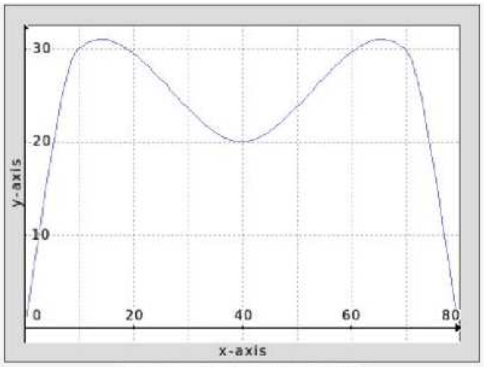
in range [0,max(xk)] (xk are positive integers).

## Example of use:

```
function1d 1,0,0,10,30,40,20,70,30,80,0 +display_graph 400,300
```



[0]: '[-1]' (81x1x1x1)



[1]: '[-1]_c1' (400x300x1x3)

# gaussian

## Arguments:

- `_sigma1[%],_sigma2[%],_angle`

## Description:

Draw a centered gaussian on selected images, with specified standard deviations and orientation.

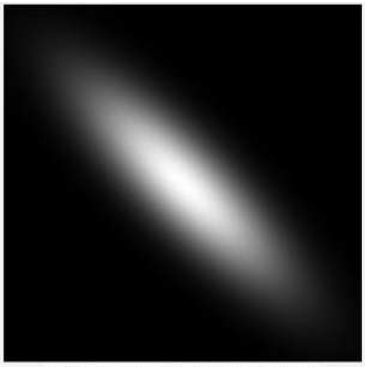
## Default values:

`sigma1=3`, `sigma2=sigma1` and `angle=0`.

This command has a **tutorial page**.

## Example of use:

```
400,400 gaussian 100,30,45
```



[0]: '[unnamed]' (400x400x1x1)

# gaussians3d

## Arguments:

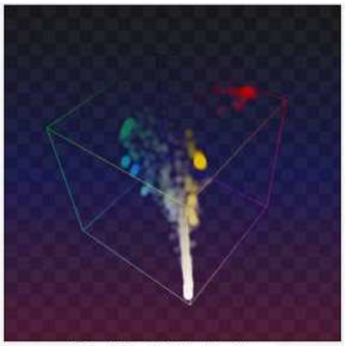
- `_size>0,_opacity`

## Description:

Convert selected 3D objects into set of 3D gaussian-shaped sprites.

## Example of use:

```
image.jpg rescale2d ,32 distribution3d gaussians3d 20 colorcube3d
primitives3d[-1] 1 +3d
```



[0]: '[3D distribution]'
(1544 vert., 1548 prim.)

# ge

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

Compute the boolean 'greater or equal than' of selected images with specified value, image

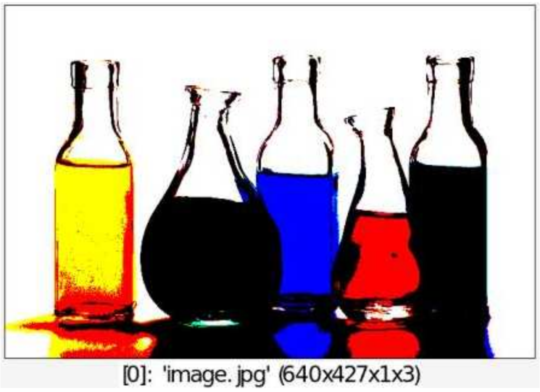
or mathematical expression, or compute the boolean 'greater or equal than' of selected images.

(*equivalent to shortcut command* `>=`).

## Examples of use:

### • Example #1

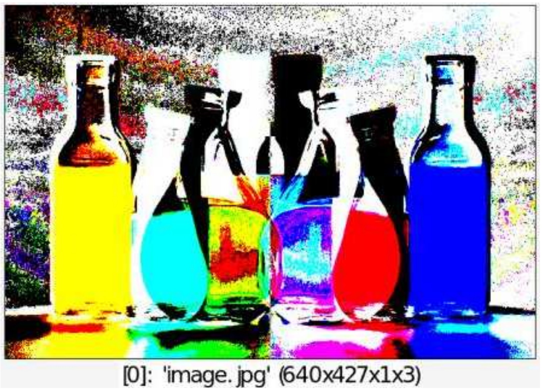
```
image.jpg ge {ia}
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +mirror x ge
```



[0]: 'image.jpg' (640x427x1x3)

---

# glow

## Arguments:

- `_amplitude>=0`

## Description:

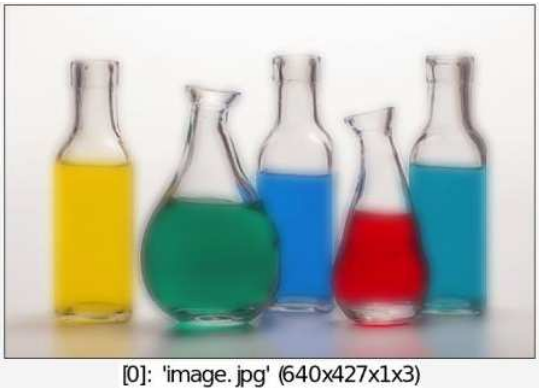
Add soft glow on selected images.

## Default values:

`amplitude=1%`.

## Example of use:

```
image.jpg glow ,
```



[0]: 'image.jpg' (640x427x1x3)

---

# gmd2ascii

## Arguments:

- `_max_line_length>0,_indent_forced_newlines>=0` or
- `(no arg)`

### Description:

Convert selected gmd-formatted text images to ascii format.

### Default values:

`max_line_length=80` and `indent_forced_newline=0`.

---

## gmd2html

### Arguments:

- `_include_default_header_footer={ 0:none | 1:Reference | 2:Tutorial | 3:News }` or
- `(no arg)`

### Description:

Convert selected gmd-formatted text images to html format.

### Default values:

`include_default_header_footer=1`.

---

## gmic3d

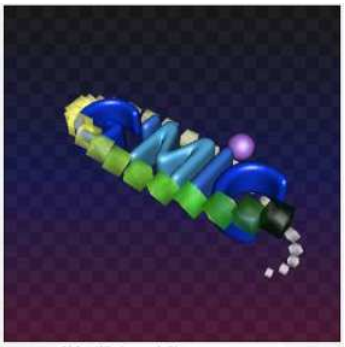
**No arguments**

### Description:

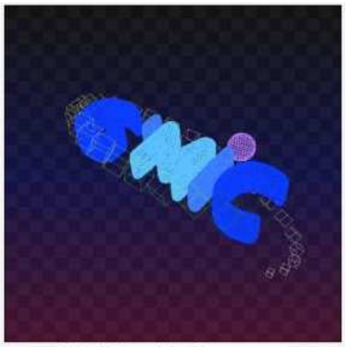
Input a 3D G'MIC logo.

### Example of use:

```
gmic3d +primitives3d 1
```



[0]: '[3d gmic]'
(21736 vert., 42764 prim.)



[1]: '[3d gmic]_c1'
(21736 vert., 64428 prim.)

# gradient

## Arguments:

- `{ x | y | z | c }...{ x | y | z | c },_scheme,_boundary_conditions` or
- `(no arg)`

## Description:

Compute the gradient components (first derivatives) of selected images, along specified axes.

(*equivalent to shortcut command* `g`).

`scheme` can be ***{ -1:backward | 0:centered | 1:forward | 2:sobel | 3:rotation-invariant (default) | 4:deriche | 5:vanvliet }***.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
(no arg) compute all significant components.

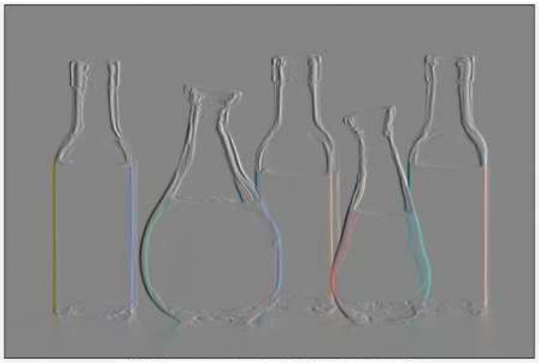
## Default values:

`scheme=0` and `boundary_conditions=1`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg gradient
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image.jpg' (640x427x1x3)

# gradient2rgb

## Arguments:

- `_is_orientation={ 0 | 1 }`

## Description:

Compute RGB representation of 2D gradient of selected images.

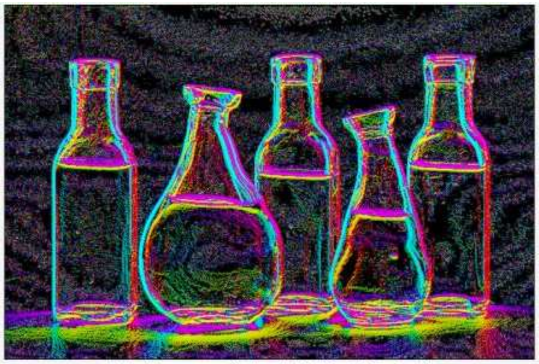
## Default values:

`is_orientation=0`.

## Example of use:

```
image.jpg +gradient2rgb 0 equalize[-1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# gradient_norm

### No arguments

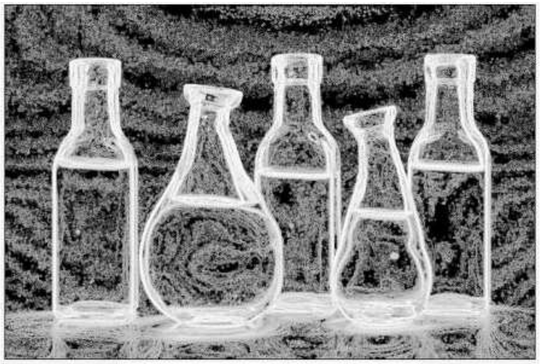
## Description:

Compute gradient norm of selected images.

This command has a **tutorial page**.

## Example of use:

```
image.jpg gradient_norm equalize
```



[0]: 'image.jpg' (640x427x1x1)

# gradient_orientation

## Arguments:

- `_dimension={ 1 | 2 | 3 }`

## Description:

Compute N-d gradient orientation of selected images.

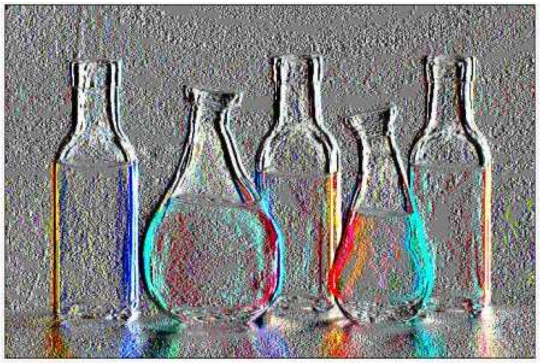
## Default values:

`dimension=3`.

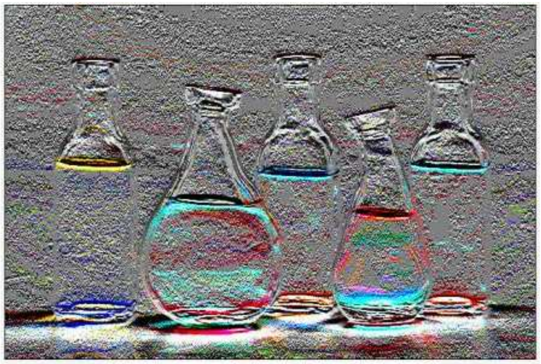
## Example of use:

```
image.jpg +gradient_orientation 2
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# graph

Built-in command

## Arguments:

- `[function_image],_plot_type,_vertex_type,_ymin,_ymax,_opacity,_pattern,_color` `...` or

- `'formula',_resolution>=0,_plot_type,_vertex_type,_xmin,xmax,_ymin,_ymax,_opac`

## Description:

Draw specified function graph on selected images.

`plot_type` can be ***{ 0:none | 1:lines | 2:splines | 3:bar }***.
`vertex_type` can be ***{ 0:none | 1:points | 2,3:crosses | 4,5:circles | 6,7:squares }***.
`pattern` is an hexadecimal number starting with `0x` which can be omitted
even if a color is specified.

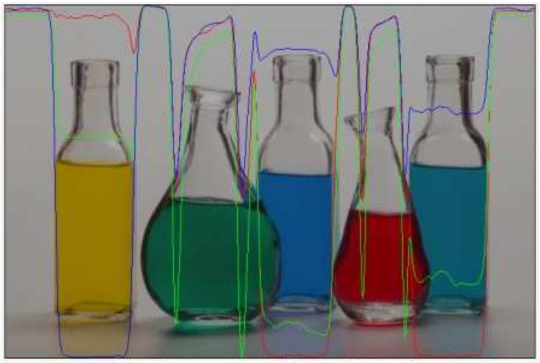
## Default values:

`plot_type=1`, `vertex_type=1`, 'ymin=ymax=0 (auto)', `opacity=1`, `pattern=(undefined)`

and `color1=0`.

## Example of use:

```
image.jpg +rows 50% blur[-1] 3 split[-1] c div[0] 1.5 graph[0] [1],
2,0,0,0,1,255,0,0 graph[0] [2],2,0,0,0,1,0,255,0 graph[0] [3],
2,0,0,0,1,0,0,255 keep[0]
```



[0]: 'image.jpg' (640x427x1x3)

# grid

## Arguments:

- `size_x[%]>=0,size_y[%]>=0,_offset_x[%],_offset_y[%],_opacity,_pattern,_color1`
  `...`

## Description:

Draw xy-grid on selected images.

`pattern` is an hexadecimal number starting with `0x` which can be omitted
even if a color is specified.

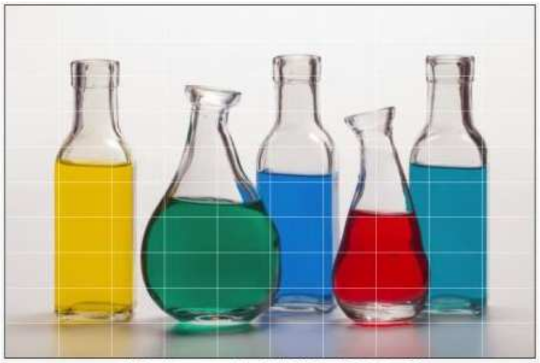
## Default values:

`offset_x=offset_y=0`, `opacity=1`, `pattern=(undefined)` and `color1=0`.

## Examples of use:

### • Example #1

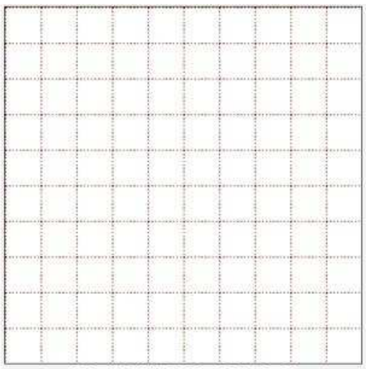
```
image.jpg grid 10%,10%,0,0,0.5,255
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
400,400,1,3,255 grid 10%,10%,0,0,0.3,0xCCCCCCCC,128,32,16
```



[0]: '[255]' (400x400x1x3)

# gt

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

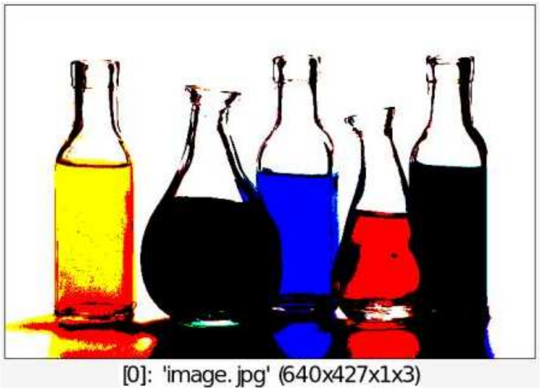
Compute the boolean 'greater than' of selected images with specified value, image or mathematical expression, or compute the boolean 'greater than' of selected images.

(*equivalent to shortcut command* `>`).

## Examples of use:

### • Example #1

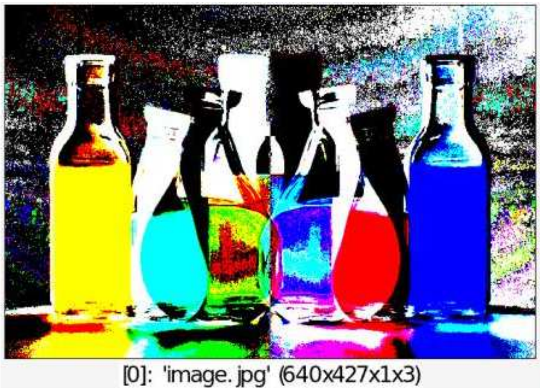
```
image.jpg gt {ia}
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +mirror x gt
```



[0]: 'image.jpg' (640x427x1x3)

---

# guided

**Built-in command**

## Arguments:

- `[guide],radius[%]>=0,regularization[%]>=0` or
- `radius[%]>=0,regularization[%]>=0`

## Description:

Blur selected images by guided image filtering.

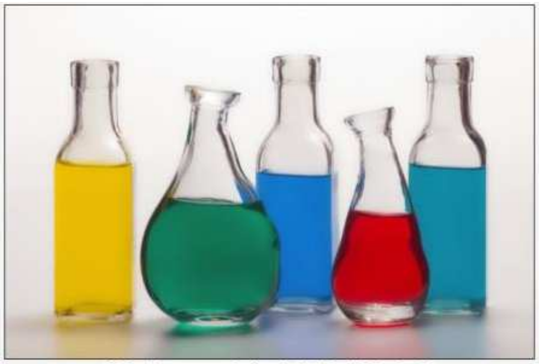
If a guide image is provided, it is used to drive the smoothing process.
A guide image must be of the same xyz-size as the selected images.

This command implements the filtering algorithm described in:
He, Kaiming; Sun, Jian; Tang, Xiaoou, "Guided Image Filtering",
IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.35, no.6, pp.1397,1409, June 2013

## Example of use:

```
image.jpg +guided 5,400
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# gyroid3d

## Arguments:

- `_resolution>0,_zoom`

## Description:

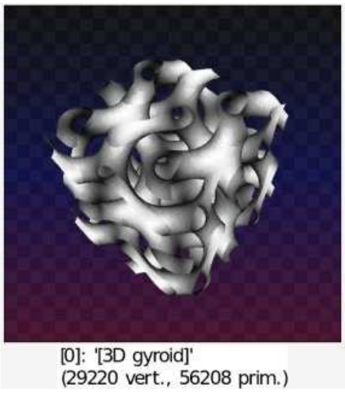
Input 3D gyroid at (0,0,0), with specified resolution.

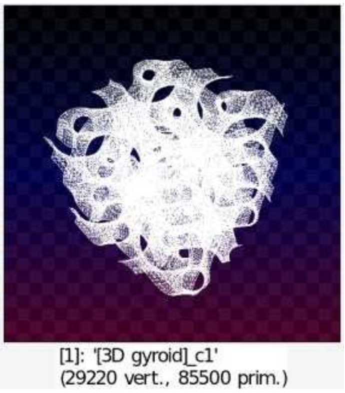
## Default values:

`resolution=32` and `zoom=5`.

## Example of use:

```
gyroid3d 48 +primitives3d 1
```


[0]: '[3D gyroid]'
(29220 vert., 56208 prim.)


[1]: '[3D gyroid]_c1'
(29220 vert., 85500 prim.)

# haar

## Arguments:

- `scale>0`

## Description:

Compute the direct haar multiscale wavelet transform of selected images.

## See also:

**ihaar** .

This command has a **tutorial page**.

# hald2clut

### No arguments

## Description:

Convert selected 2D HaldCLUTs to 3D CLUTs.

# halftone

## Arguments:

- `nb_levels>=2,_size_dark>=2,_size_bright>=2,_shape={ 0:square | 1:diamond | 2:circle | 3:inv-square | 4:inv-diamond | 5:inv-circle },_smoothness[%]>=0`

## Description:

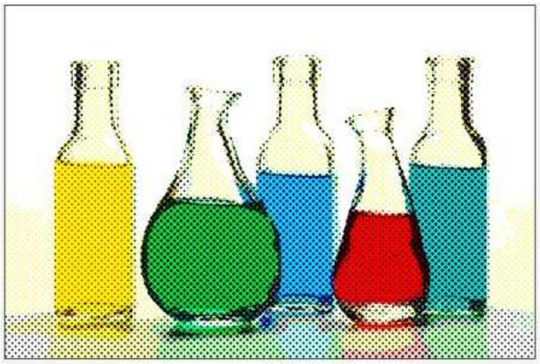
Apply halftone dithering to selected images.

### Default values:

`nb_levels=5`, `size_dark=8`, `size_bright=8`, `shape=5` and `smoothnessss=0`.

### Example of use:

```
image.jpg halftone ,
```



[0]: 'image.jpg' (640x427x1x3)

## hardsketchbw

### Arguments:

- `_amplitude>=0,_density>=0,_opacity,0<=_edge_threshold<=100,_is_fast={ 0 | 1 }`

### Description:

Apply hard B&W sketch effect on selected images.

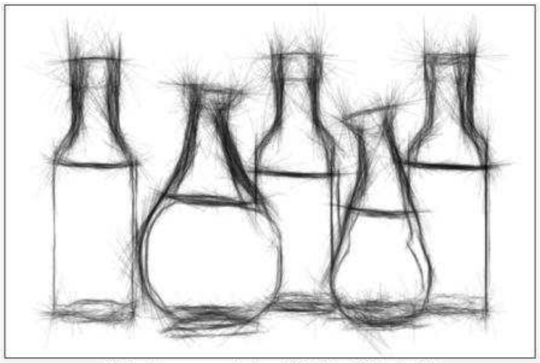
### Default values:

`amplitude=1000`, `sampling=3`, `opacity=0.1`, `edge_threshold=20` and `is_fast=0`.

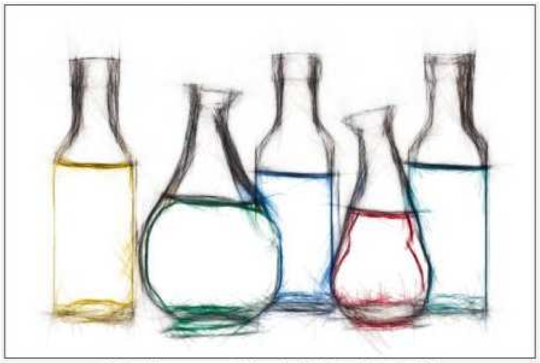
### Example of use:

```
image.jpg +hardsketchbw 200,70,0.1,10 median[-1] 2 +local reverse
blur[-1] 3 blend[-2,-1] overlay done
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x3)

---

## hcy2rgb

**No arguments**

### Description:

Convert color representation of selected images from HCY to RGB.

---

## hearts

### Arguments:

- `_density>=0`

### Description:

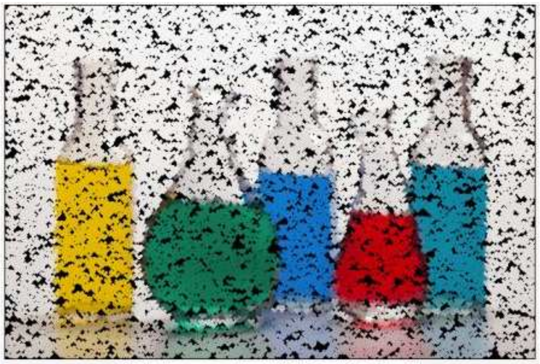
Apply heart effect on selected images.

### Default values:

`density=10`.

### Example of use:

```
image.jpg hearts ,
```



[0]: 'image.jpg' (640x427x1x3)

# heat_flow

## Arguments:

- `_nb_iter>=0,_dt,_keep_sequence={ 0 | 1 }`

## Description:

Apply iterations of the heat flow on selected images.

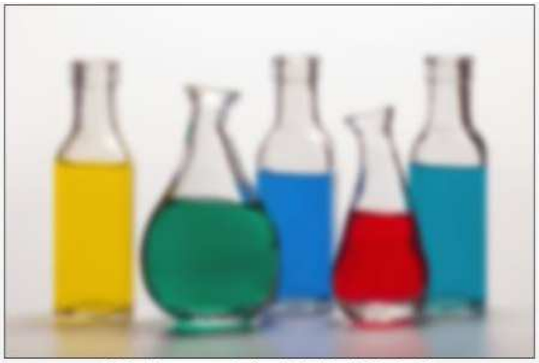
## Default values:

`nb_iter=10`, `dt=30` and `keep_sequence=0`.

## Example of use:

```
image.jpg +heat_flow 20
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# help

## Arguments:

- `command` or

- `(no arg)`

## Description:

Display help (optionally for specified command only) and exit.

(*equivalent to shortcut command* `h`).

---

# hessian

## Arguments:

- `{ xx | xy | xz | yy | yz | zz }...{ xx | xy | xz | yy | yz | zz },_boundary_conditions` or
- `(no arg) :`

## Description:

Compute the hessian components (second derivatives) of selected images along specified axes.

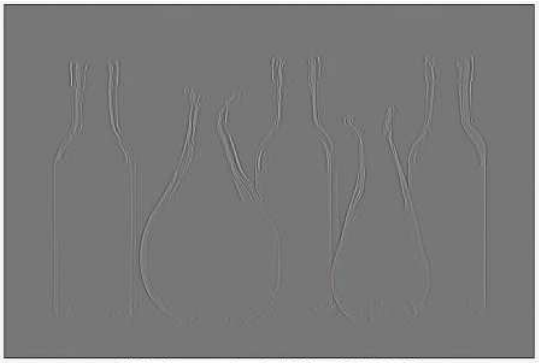
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
(no arg) compute all significant components.

## Default values:

`boundary_conditions=1`.

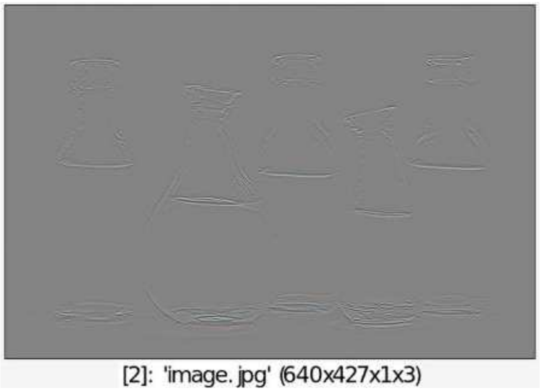
## Example of use:

```
image.jpg hessian
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image.jpg' (640x427x1x3)



[2]: 'image.jpg' (640x427x1x3)

# hex

## Arguments:

- `hexadecimal_int1,...`

## Description:

Print specified hexadecimal integers into their binary, octal, decimal and string representations.

# hex2dec

## Arguments:

- `hexadecimal_int1,...`

## Description:

Convert specified hexadecimal integers into their decimal representations.

# hex2img

## Arguments:

- `"hexadecimal_string"`

## Description:

Insert new image 1xN at the end of the list with values specified by the given hexadecimal-encoded string.

# hex2str

## Arguments:

- `hexadecimal_string`

## Description:

Convert specified hexadecimal string into a string.

## See also:

**str2hex**.

# histogram

**Built-in command**

## Arguments:

- `nb_levels>0[%],_min_value[%],_max_value[%]`

## Description:

Compute the histogram of selected images.

If value range is set, the histogram is estimated only for pixels in the specified value range. Argument `max_value` must be specified if `min_value` is set.

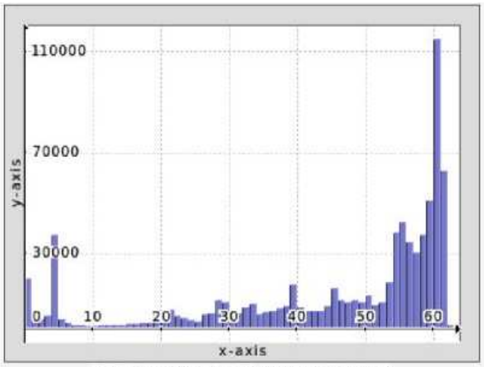
## Default values:

`min_value=0%` and `max_value=100%`.

## Example of use:

```
image.jpg +histogram 64 display_graph[-1] 400,300,3
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (400x300x1x3)

# histogram3d

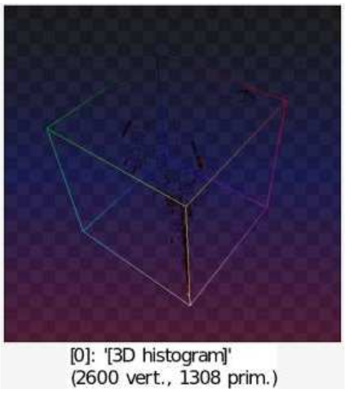
**No arguments**

## Description:

Get 3D color histogram of selected images.

## Example of use:

```
image.jpg rescale2d 64 histogram3d circles3d 3 opacity3d. 0.75
colorcube3d primitives3d[-1] 1 add3d
```



[0]: '[3D histogram]'
(2600 vert., 1308 prim.)

# histogram_cumul

## Arguments:

- `_nb_levels>0,_is_normalized={ 0 | 1 },_val0[%],_val1[%]`

## Description:

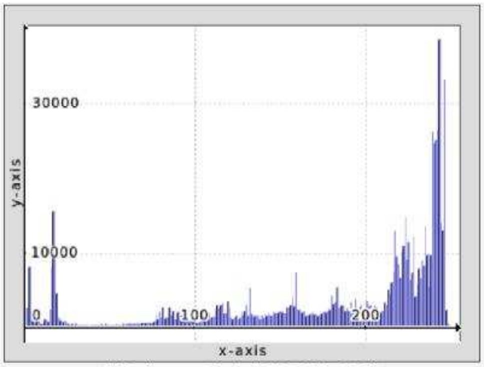
Compute cumulative histogram of selected images.

## Default values:

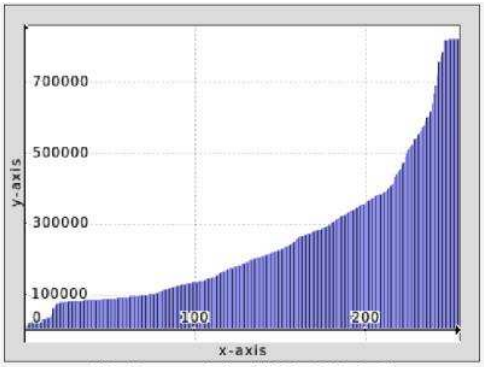
`nb_levels=256`, `is_normalized=0`, `val0=0%` and `val1=100%`.

## Example of use:

```
image.jpg +histogram_cumul 256 histogram[0] 256 display_graph
400,300,3
```



[0]: 'image.jpg' (400x300x1x3)



[1]: 'image_c1.jpg' (400x300x1x3)

# histogram_masked

## Arguments:

- `[mask],nb_levels>0[%],_min_value[%],_max_value[%]`

## Description:

Compute the masked histogram of selected images.

## Default values:

`min_value=0%` and `max_value=100%`.

# histogram_nd

## Arguments:

- `nb_levels>0[%],_value0[%],_value1[%]`

## Description:

Compute the 1D,2D or 3D histogram of selected multi-channels images (having 1,2 or 3 channels).

If value range is set, the histogram is estimated only for pixels in the specified
value range.

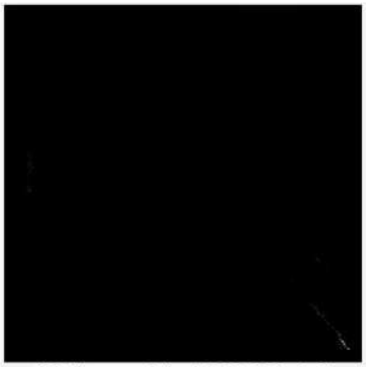
## Default values:

`value0=0%` and `value1=100%`.

## Example of use:

```
image.jpg channels 0,1 +histogram_nd 256
```

[0]: 'image.jpg' (640x427x1x2)



[1]: 'image_c1.jpg' (256x256x1x1)

# histogram_pointwise

## Arguments:

- `nb_levels>0[%],_value0[%],_value1[%]`

## Description:

Compute the histogram of each vector-valued point of selected images.

If value range is set, the histogram is estimated only for values in the specified value range.

## Default values:

`value0=0%` and `value1=100%`.

# hough

## Arguments:

- `_width>0,_height>0,gradient_norm_voting={ 0 | 1 }`

## Description:

Compute hough transform (theta,rho) of selected images.

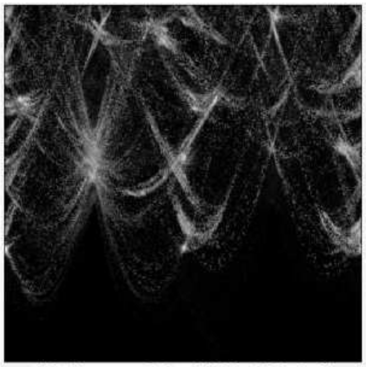
## Default values:

`width=512`, `height=width` and `gradient_norm_voting=1`.

## Example of use:

```
image.jpg +blur 1.5 hough[-1] 400,400 blur[-1] 0.5 add[-1] 1 log[-1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (400x400x1x1)

# houghsketchbw

## Arguments:

- `_density>=0,_radius>0,0<=_threshold<=100,0<=_opacity<=1,_votesize[%]>0`

## Description:

Apply hough B&W sketch effect on selected images.

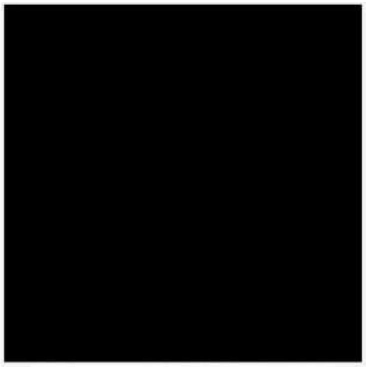
## Default values:

`density=100`, `radius=3`, `threshold=100`, `opacity=0.1` and `votesize=100%`.

## Example of use:

```
image.jpg +houghsketchbw ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x640x1x1)

# hsi2rgb

**No arguments**

## Description:

Convert color representation of selected images from HSI to RGB.

# hsi82rgb

**No arguments**

## Description:

Convert color representation of selected images from HSI8 to RGB.

# hsl2rgb

**No arguments**

## Description:

Convert color representation of selected images from HSL to RGB.

# hsl82rgb

**No arguments**

## Description:

Convert color representation of selected images from HSL8 to RGB.

# hsv2rgb

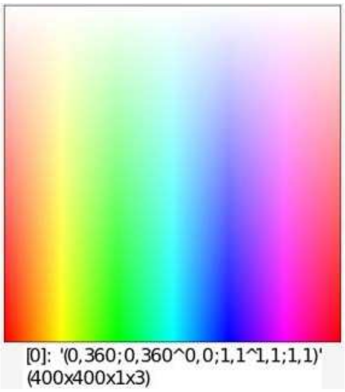
**No arguments**

## Description:

Convert color representation of selected images from HSV to RGB.

## Example of use:

```
(0,360;0,360^0,0;1,1^1,1;1,1) resize 400,400,1,3,3 hsv2rgb
```



[0]: '(0,360;0,360^0,0;1,1^1,1;1,1)'
(400x400x1x3)

---

# hsv82rgb

### No arguments

## Description:

Convert color representation of selected images from HSV8 to RGB.

---

# huffman_tree

### No arguments

## Description:

Generate Huffman coding tree from the statistics of all selected images.

Huffman tree is returned as a 1xN image inserted at the end of the image list, representing the `N` vector-valued leafs/nodes of the tree, encoded as `[ value,parent,child0,child1 ]`.
Last row of the returned image corresponds to the tree root.
Selected images must contain only positive integer values.
Return maximal value of the input data in the status.

## See also:

**compress_huffman**, **decompress_huffman**.

---

# idct

## Arguments:

- `_{ x | y | z }...{ x | y | z }` or
- `(no arg)`

## Description:

Compute the inverse discrete cosine transform of selected images, optionally along the specified axes only.

Output images are always evenly sized, so this command may change the size of the selected images.
(dct images obtained with the `dct` command are evenly sized anyway).

### Default values:

(no arg)

### See also:

**dct**.

This command has a **tutorial page**.

---

## identity

### Arguments:

- `_width>=0,_height>=0,_depth>=0`

### Description:

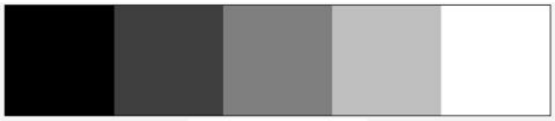
Insert an identity map of given size at the end of the image list.

### Default values:

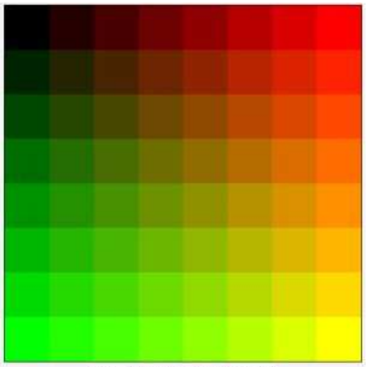
`height=width` and `depth=1`.

### Example of use:

```
identity 5,1 identity 8,8
```



[0]: '[x]' (5x1x1x1)



[1]: '[[x,y]]' (8x8x1x2)

---

# iee

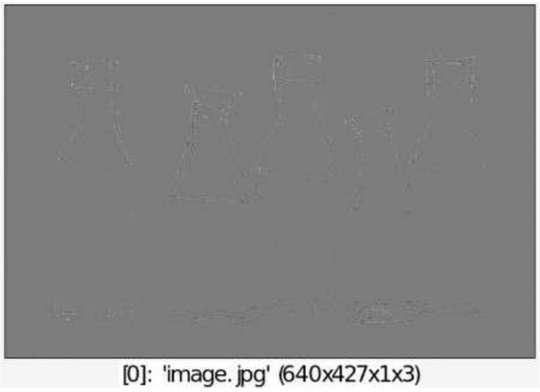
**No arguments**

## Description:

Compute gradient-orthogonal-directed 2nd derivative of image(s).

## Example of use:

```
image.jpg iee
```



[0]: 'image.jpg' (640x427x1x3)

# if

**Built-in command**

## Arguments:

- `condition`

## Description:

Start a `if...[elif]...[else]...fi` block and test if specified condition holds.

`condition` is a mathematical expression, whose evaluation is interpreted as ***`{ 0:false | other:true }`***.

This command has a **tutorial page**.

## Example of use:

```
image.jpg if ia<64 add 50% elif ia<128 add 25% elif ia<192 sub 25%
else sub 50% fi cut 0,255
```

[0]: 'image.jpg' (640x427x1x3)

# ifft

Built-in command

## Arguments:

- `_{ x | y | z }...{ x | y | z }`

## Description:

Compute the inverse fourier transform (real and imaginary parts) of selected images.

optionally along the specified axes only.

## See also:

**fft**.

This command has a **tutorial page**.

# ifftpolar

**No arguments**

## Description:

Compute inverse fourier transform of selected images, from centered magnitude/phase images.

# ihaar

## Arguments:

- `scale>0`

## Description:

Compute the inverse haar multiscale wavelet transform of selected images.

### See also:

**haar** .

This command has a **tutorial page**.

---

## ilaplacian

### Arguments:

- `{ nb_iterations>0 | 0 },_[initial_estimate]`

### Description:

Invert selected Laplacian images.

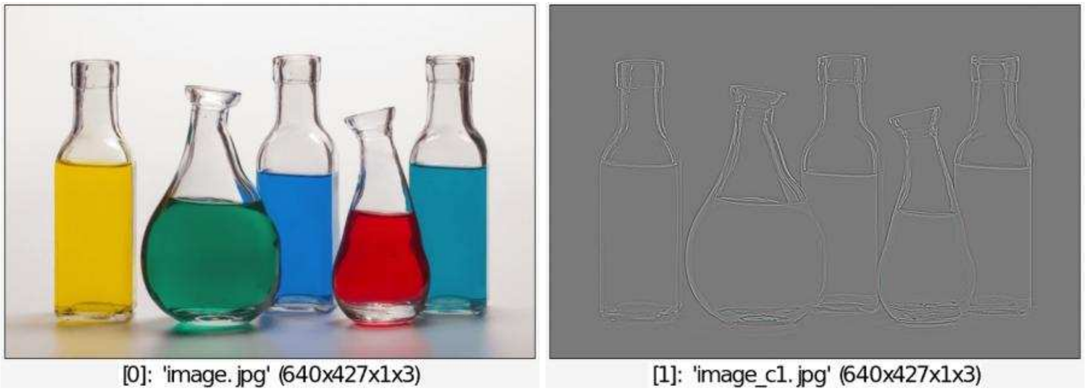
If given `nb_iterations` is `0`, inversion is done in Fourier space (single iteration),
otherwise, by applying `nb_iterations` of a Laplacian-inversion PDE flow.
Note that the resulting inversions are just estimation of possible/approximated solutions.

### Default values:

`nb_iterations=0` and `[initial_estimated]=(undefined)`.

### Example of use:

```
image.jpg +laplacian +ilaplacian[-1] 0
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c2.jpg' (640x427x1x3)

## image

Built-in command

### Arguments:

- `[sprite],_x[%|~],_y[%|~],_z[%|~],_c[%|~],_opacity,_[opacity_mask],_max_opacity_mask`

### Description:

Draw specified sprite on selected images.

(*equivalent to shortcut command* `j`).

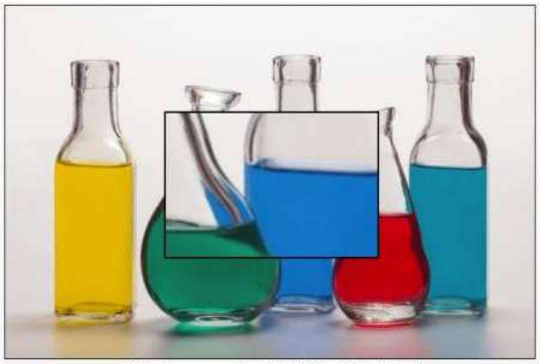
If one of the x,y,z or c argument ends with a `~`, its value is expected to be
a centering ratio (in [0,1]) rather than a position.
Usual centering ratio are ***`{ 0:left-justified | 0.5:centered | 1:right-justified }`***.

### Default values:

`x=y=z=c=0`, `opacity=1`, `opacity_mask=(undefined)` and `max_opacity_mask=1`.

### Example of use:

```
image.jpg +crop 40%,40%,60%,60% resize[-1] 200%,200%,1,3,5 frame[-1]
2,2,0 image[0] [-1],30%,30% keep[0]
```



[0]: 'image.jpg' (640x427x1x3)

# image6cube3d

**No arguments**

## Description:

Generate 3D mapped cubes from 6-sets of selected images.

## Example of use:

```
image.jpg animate flower,"30,0","30,5",6 image6cube3d
```

[0]: '[3D image cube]' (24 vert., 6 prim.)

# imagealpha

## Arguments:

- `[sprite],_x[%|~],_y[%|~],_z[%|~],_c[%|~],_opacity`

## Description:

Draw specified sprite on selected images, considering that the sprite's last channel is the drawing's alpha.

(*equivalent to shortcut command* `ja`).

If one of the x,y,z or c argument ends with a `~`, its value is expected to be
a centering ratio (in [0,1]) rather than a position.
Usual centering ratio are ***`{ 0:left-justified | 0.5:centered | 1:right-justified }`***.

## Default values:

`x=y=z=c=0` and `opacity=1`.

# imageblocks3d

## Arguments:

- `_maximum_elevation,_smoothness[%]>=0`

## Description:

Generate 3D blocks from selected images.

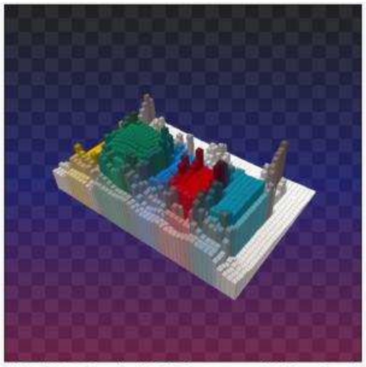
Transparency of selected images is taken into account.

## Default values:

`maximum_elevation=10` and `smoothness=0`.

## Example of use:

```
image.jpg rescale2d ,32 imageblocks3d -20 mode3d 3
```



[0]: '[3D box]' (12288 vert., 9216 prim.)

---

# imagecube3d

**No arguments**

## Description:

Generate 3D mapped cubes from selected images.

## Example of use:

```
image.jpg imagecube3d
```

[0]: 'image.jpg' (8 vert., 6 prim.)

---

# imagegrid

## Arguments:

- `M>0,_N>0`

## Description:

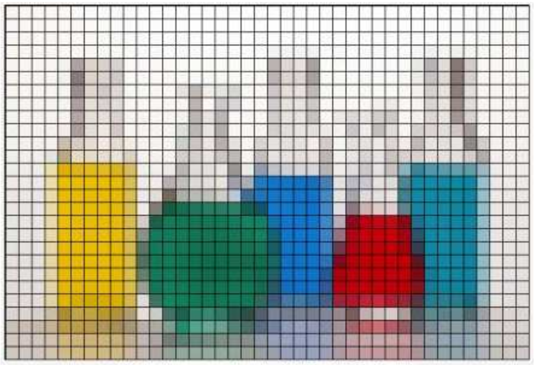
Create MxN image grid from selected images.

## Default values:

`N=M`.

## Example of use:

```
image.jpg imagegrid 16
```



[0]: 'image.jpg' (640x432x1x3)

---

# imagegrid_hexagonal

## Arguments:

- `_resolution>0,0<=_outline<=1`

## Description:

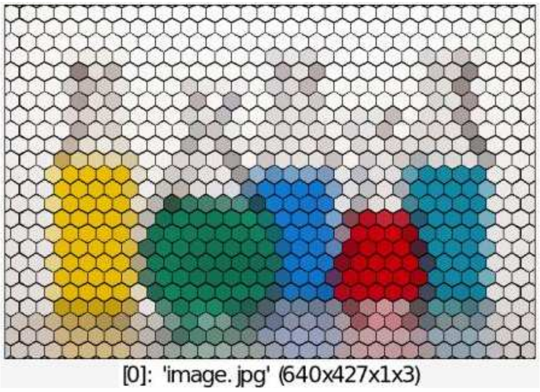
Create hexagonal grids from selected images.

## Default values:

`resolution=32`, `outline=0.1` and `is_antialiased=1`.

## Example of use:

```
image.jpg imagegrid_hexagonal 24
```



[0]: 'image.jpg' (640x427x1x3)

# imagegrid_triangular

## Arguments:

- `pattern_width>=1,_pattern_height>=1,_pattern_type,0<=_outline_opacity<=1,_out`

## Description:

Create triangular grids from selected images.

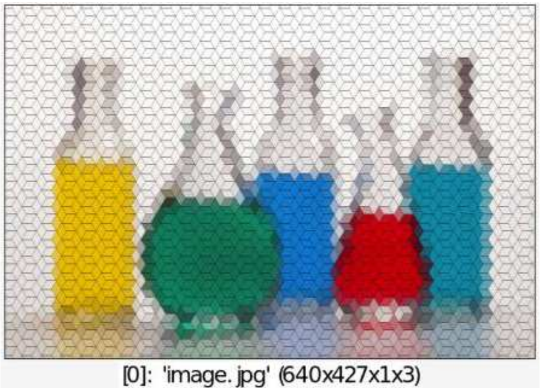
'pattern type' can be ***{ 0:horizontal | 1:vertical | 2:crossed | 3:cube | 4:decreasing | 5:increasing }***.

## Default values:

`pattern_width=24`, `pattern_height=pattern_width`, `pattern_type=0`, `outline_opacity=0.1` and `outline_color1=0`.

## Example of use:

```
image.jpg imagegrid_triangular 6,10,3,0.5
```



[0]: 'image.jpg' (640x427x1x3)

# imageplane3d

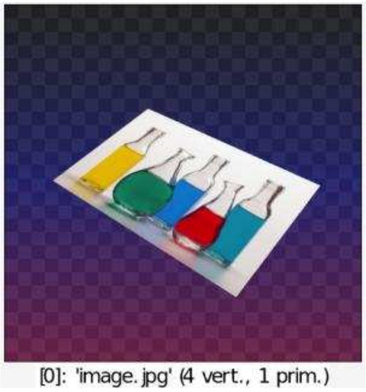
**No arguments**

## Description:

Generate 3D mapped planes from selected images.

## Example of use:

```
image.jpg imageplane3d
```



[0]: 'image.jpg' (4 vert., 1 prim.)

# imagepyramid3d

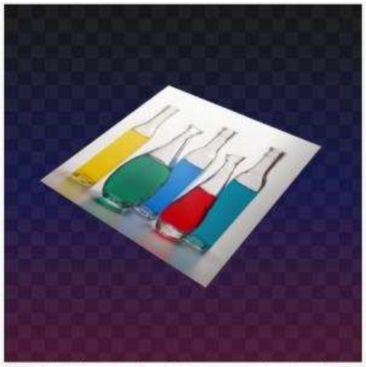
**No arguments**

## Description:

Generate 3D mapped pyramids from selected images.

## Example of use:

```
image.jpg imagepyramid3d
```



[0]: 'image.jpg' (5 vert., 5 prim.)

## imagerubik3d

### Arguments:

- `_xy_tiles>=1,0<=xy_shift<=100,0<=z_shift<=100`

### Description:

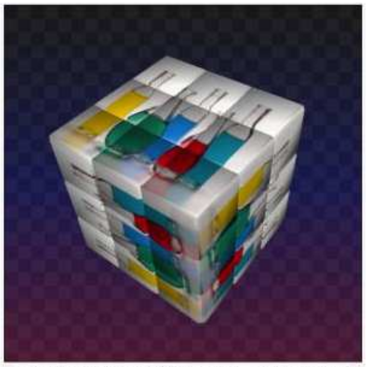
Generate 3D mapped rubik's cubes from selected images.

### Default values:

`xy_tiles=3`, `xy_shift=5` and `z_shift=5`.

### Example of use:

```
image.jpg imagerubik3d ,
```



[0]: 'image.jpg' (432 vert., 270 prim.)

## imagesphere3d

### Arguments:

- `_resolution1>=3,_resolution2>=3`

## Description:

Generate 3D mapped sphere from selected images.

## Default values:

`resolution1=32` and `resolutions2=16`.

## Example of use:

```
image.jpg imagesphere3d 32,16
```

[0]: 'image.jpg' (450 vert., 480 prim.)

---

# img2ascii

## Arguments:

- `_charset,_analysis_scale>0,_analysis_smoothness[%]>=0,_synthesis_scale>0,_out`

## Description:

Render selected images as binary ascii art.

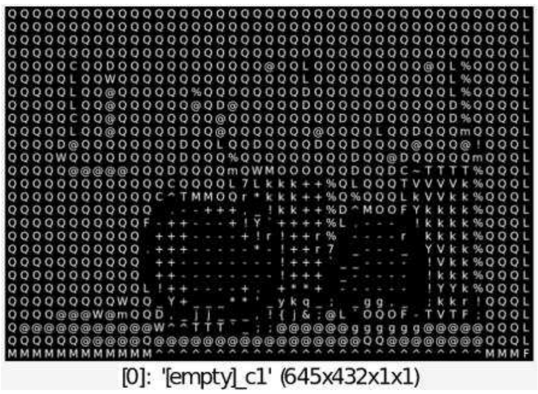
This command returns the corresponding the list of widths and heights (expressed as a number of characters)
for each selected image.

## Default values:

'charset=[ascii charset]', `analysis_scale=16`, `analysis_smoothness=20%`, `synthesis_scale=16` and `_output_ascii_filename=[undefined]`.

## Example of use:

```
image.jpg img2ascii ,
```



[0]: '[empty]_c1' (645x432x1x1)

# img2base64

## Arguments:

- `_encoding={ 0:base64 | 1:base64url },_store_names={ 0 | 1 }`

## Description:

Encode selected images as a base64-encoded string.

The images can be then decoded using command `base642img`.

## Default values:

`encoding=0` and `store_names=1`.

# img2hex

**No arguments**

## Description:

Return representation of last image as an hexadecimal-encoded string.

Input image must have values that are integers in [0,255].

# img2patches

## Arguments:

- `patch_size>0,_overlap[%]>0,_boundary_conditions`

## Description:

Decompose selected 2D images into (possibly overlapping) patches and stack them along the z-axis.

`overlap` must be in range `[0,patch_size-1]`.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

### Default values:

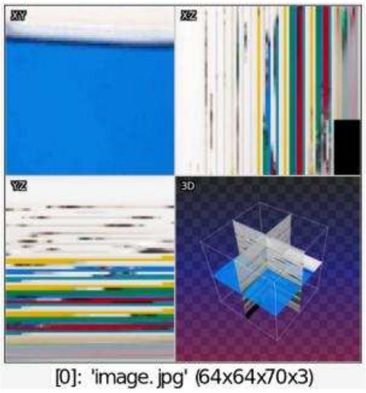
`overlap=0` and `boundary_conditions=0`.

### See also:

**patches2img**.

### Example of use:

```
image.jpg img2patches 64
```



[0]: 'image.jpg' (64x64x70x3)

## img2str

**No arguments**

### Description:

Return the content of the selected images, as special G'MIC input strings.

## img2text

### Arguments:

- `_line_separator`

### Description:

Return text contained in a multi-line image.

### Default values:

'line_separator= '.

# Preamble

- This document is distributed under the **GNU Free Documentation License**, version 1.3.
- A **.pdf version** of this document is available.
- Quick access to the **List of Commands**.

# Version

**G'MIC**: `GREYC's Magic for Image Computing`
**https://gmic.eu**
Version **3.3.5**

# Table of Contents

---

# inn

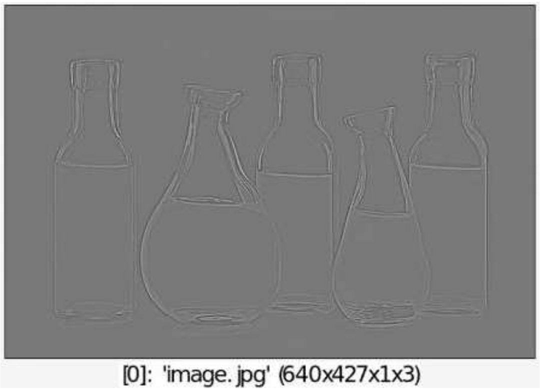
### No arguments

### Description:

Compute gradient-directed 2nd derivative of image(s).

## Example of use:

```
image.jpg inn
```


[0]: 'image.jpg' (640x427x1x3)

# inpaint

Built-in command

## Arguments:

- `[mask]` or
- `[mask],0,_fast_method` or
- `[mask],_patch_size>=1,_lookup_size>=1,_lookup_factor>=0,_lookup_increment!=0,_blend_size>=0,0<=_blend_threshold<=1,_blend_decay>=0,_blend_scales>=1,_is 0 | 1 }`

## Description:

Inpaint selected images by specified mask.

If no patch size (or 0) is specified, inpainting is done using a fast average or median algorithm. Otherwise, it used a patch-based reconstruction method, that can be very time consuming. `fast_method` can be ***{ 0:low-connectivity average | 1:high-connectivity average | 2:low-connectivity median | 3:high-connectivity median }***.

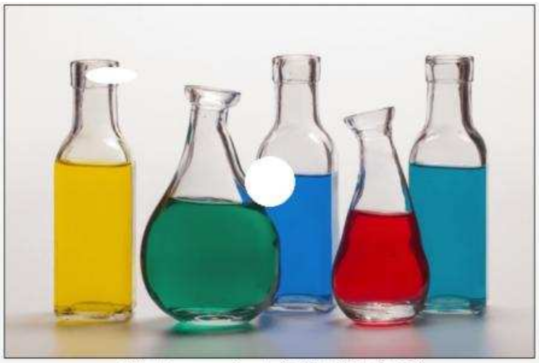
## Default values:

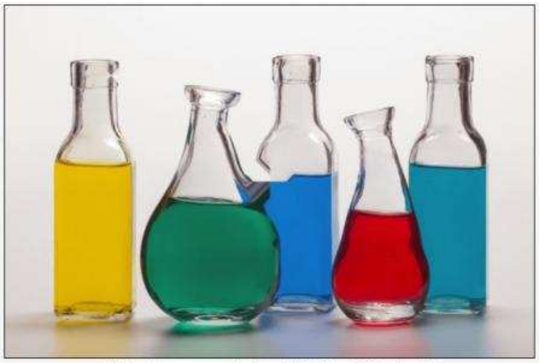
`patch_size=0`, `fast_method=1`, `lookup_size=22`, `lookup_factor=0.5`, `lookup_increment=1`, `blend_size=0`, `blend_threshold=0`, `blend_decay=0.05`, `blend_scales=10` and `is_blend_outer=1`.

## Examples of use:

### • Example #1

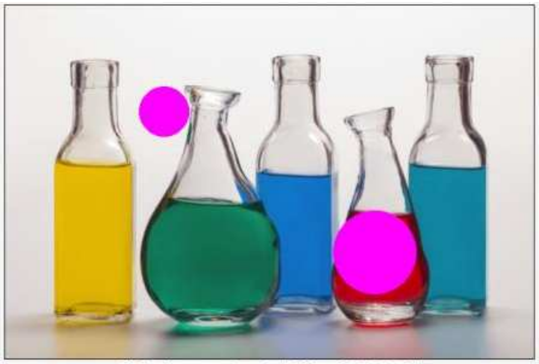
```
image.jpg 100%,100% ellipse 50%,50%,30,30,0,1,255 ellipse 20%,20%,
30,10,0,1,255 +inpaint[-2] [-1] remove[-2]
```

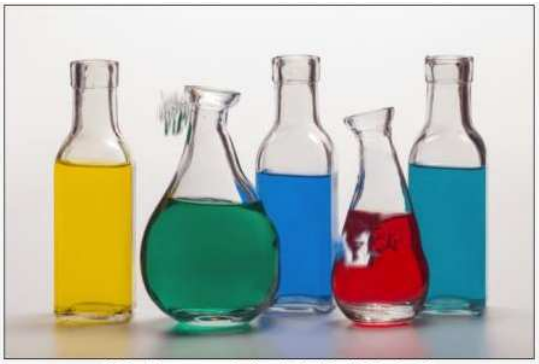

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg 100%,100% circle 30%,30%,30,1,255,0,255 circle 70%,70%,
50,1,255,0,255 +inpaint[0] [1],5,15,0.5,1,9,0 remove[1]
```


[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

# inpaint_flow

## Arguments:

- `[mask],_nb_global_iter>=0,_nb_local_iter>=0,_dt>0,_alpha>=0,_sigma>=0`

## Description:

Apply iteration of the inpainting flow on selected images.

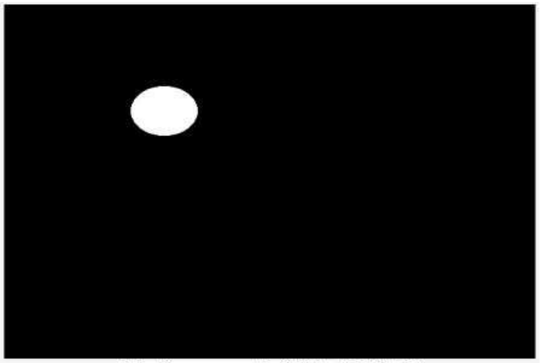
## Default values:

`nb_global_iter=10`, `nb_local_iter=100`, `dt=5`, `alpha=1` and `sigma=3`.

## Example of use:

```
image.jpg 100%,100% ellipse[-1] 30%,30%,40,30,0,1,255 inpaint_flow[0]
[1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[unnamed]' (640x427x1x1)

# inpaint_holes

## Arguments:

- `maximal_area[%]>=0,_tolerance>=0,_is_high_connectivity={ 0 | 1 }`

## Description:

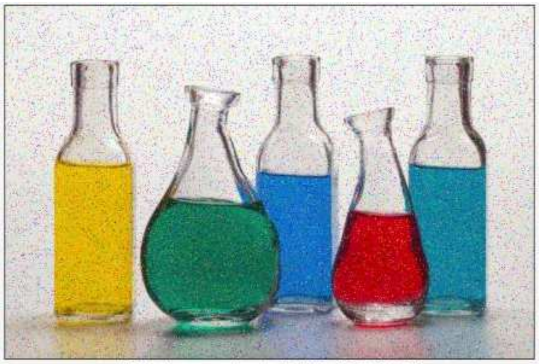
Inpaint all connected regions having an area less than specified value.

## Default values:

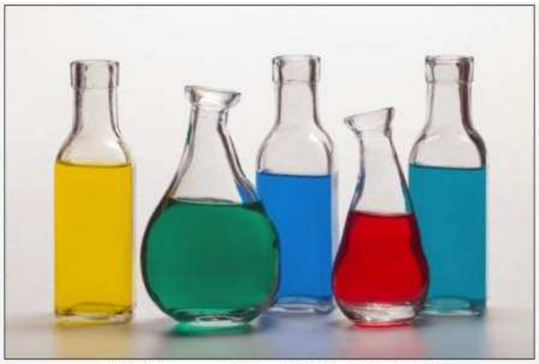
`maximal_area=4`, `tolerance=0` and `is_high_connectivity=0`.

## Example of use:

```
image.jpg noise 5%,2 +inpaint_holes 8,40
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# inpaint_matchpatch

## Arguments:

- `[mask],_nb_scales={ 0:auto | >0 },_patch_size>0,_nb_iterations_per_scale>0,_blend_size>=0,_allow_outer_blendi`

`0 | 1 },_is_already_initialized={ 0 | 1 }`

## Description:

Inpaint selected images by specified binary mask, using a multi-scale matchpatch algorithm.

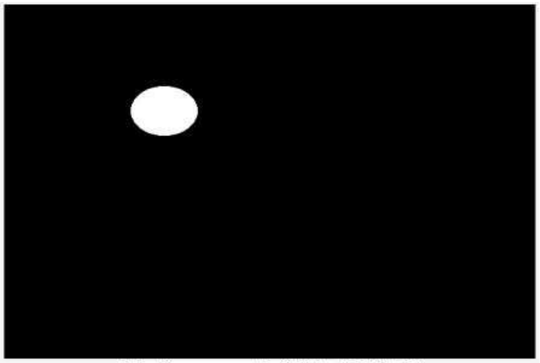
## Default values:

`nb_scales=0`, `patch_size=9`, `nb_iterations_per_scale=10`, `blend_size=5`, `allow_outer_blending=1` and `is_already_initialized=0`.

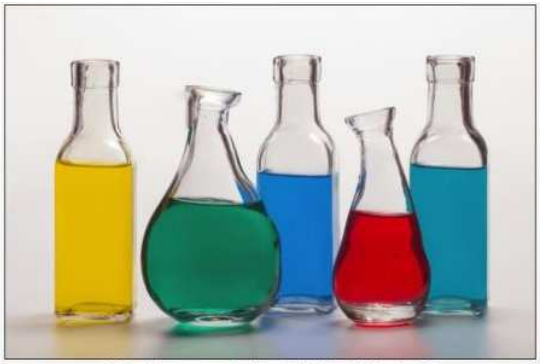
## Example of use:

```
image.jpg 100%,100% ellipse[-1] 30%,30%,40,30,0,1,255
+inpaint_matchpatch[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[unnamed]' (640x427x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

# inpaint_morpho

## Arguments:

- `[mask]`

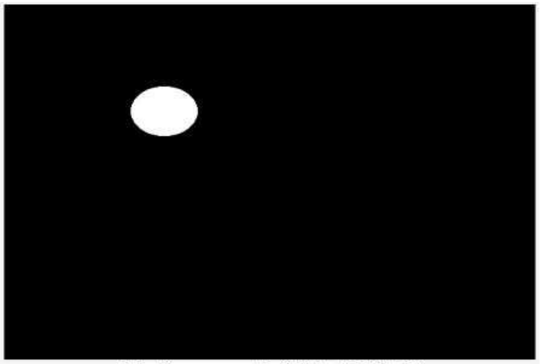
## Description:

Inpaint selected images by specified mask using morphological operators.

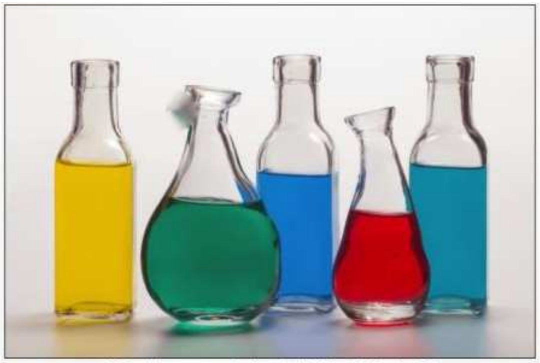
## Example of use:

```
image.jpg 100%,100% ellipse[-1] 30%,30%,40,30,0,1,255
+inpaint_morpho[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[unnamed]' (640x427x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

# inpaint_pde

## Arguments:

- `[mask],_nb_scales[%],_diffusion_type={ 0:isotropic | 1:Delaunay-guided | 2:edge-guided | 3:mask-guided },_diffusion_iter>=0`

## Description:

Inpaint selected images by specified mask using a multiscale transport-diffusion algorithm.

Argument `nb_scales` sets the number of scales used in the multi-scale resolution scheme.

- When the `%` qualifier is used for `nb_scales`, the number of used scales is relative to `nb_scales_max = ceil(log2(max(w,h,d)))`.
- When `nb_scales<0`, it determines the minimum image size encountered at the lowest scale.

If `diffusion_type==3`, non-zero values of the mask (e.g. a distance function) are used to guide the diffusion process.

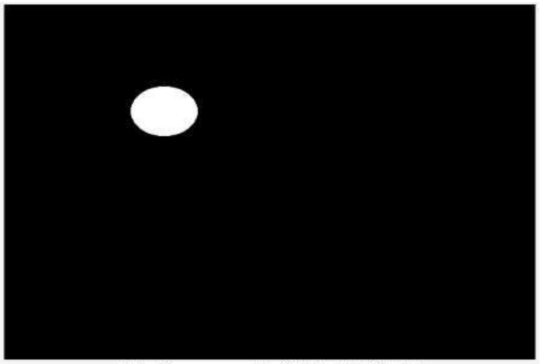
## Default values:

`nb_scales=-9`, `diffusion_type=1` and `diffusion_iter=20`.

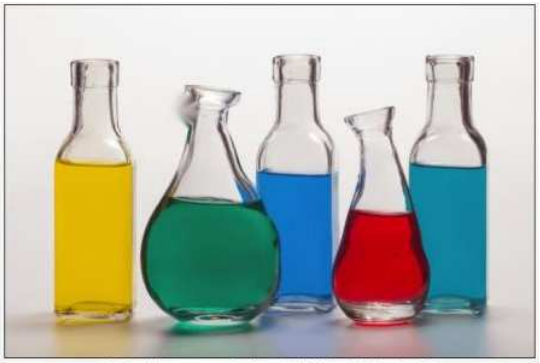
## Example of use:

```
image.jpg 100%,100% ellipse[-1] 30%,30%,40,30,0,1,255 +inpaint_pde[0]
[1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[unnamed]' (640x427x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

# input

Built-in command

## Arguments:

- `[type:]filename` or
- `[type:]http://URL` or
- `[selection]x_nb_copies>0` or
- `{ width>0[%] | [image_w] },{ _height>0[%] | [image_h] },{ _depth>0[%] | [image_d] },{ _spectrum>0[%] | [image_s] },_{ value1,_value2,... | 'formula' }` or
- `(value1{,|;|/|^}value2{,|;|/|^}...[:{x|y|z|c|,|;|/|^}])` or
- `0`

## Description:

Insert a new image taken from a filename or from a copy of an existing image [index],

or insert new image with specified dimensions and values. Single quotes may be omitted in `formula`. Specifying argument `0` inserts an `empty` image.

(*equivalent to shortcut command* `i`).

## Default values:

`nb_copies=1`, `height=depth=spectrum=1` and `value1=0`.

This command has a **tutorial page**.

## Examples of use:

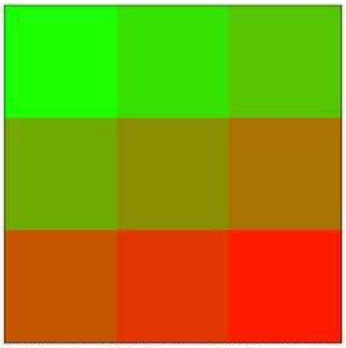
### • Example #1

```
input image.jpg
```

[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
input (1,2,3;4,5,6;7,8,9^9,8,7;6,5,4;3,2,1)
```

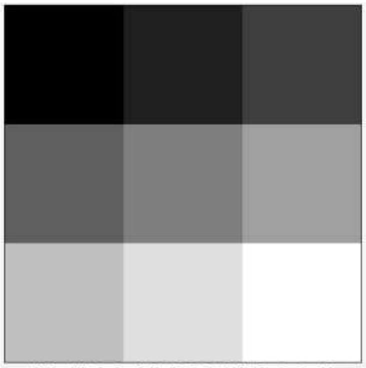


[0]: '(1,2,3;4,5,6;7,8,9^9,8,7;6,5,4;3,2,1)' (3x3x1x2)

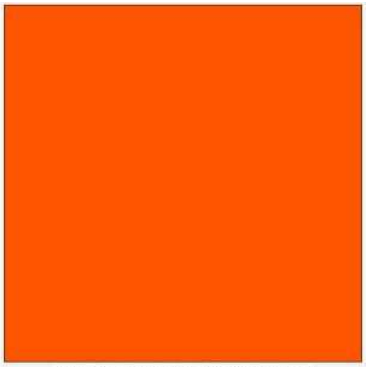
### • Example #3

```
image.jpg (1,2,3;4,5,6;7,8,9) (255^128^64) 400,400,1,3,'(x>w/2?
x:y)*c'
```

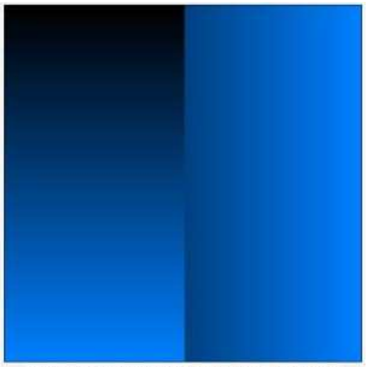
[0]: 'image.jpg' (640x427x1x3)



[1]: '(1,2,3;4,5,6;7,8,9)' (3x3x1x1)



[2]: '(255^128^64)' (1x1x1x3)



[3]: '[(x>w/2?x:y)*c]' (400x400x1x3)

# input_565

## Arguments:

- `filename,width>0,height>0,reverse_endianness={ 0 | 1 }`

## Description:

Insert image data from a raw RGB-565 file, at the end of the list.

## Default values:

`reverse_endianness=0`.

# input_bytes

## Arguments:

- `filename`

## Description:

Input specified filename as a 1D array of bytes.

(*equivalent to shortcut command* `ib`).

# input_cached

## Arguments:

- `"basename.ext",_try_downloading_from_gmic_server={ 0 | 1 }`

## Description:

Input specified filename, assumed to be stored in one of the G'MIC resource folder.

If file not found and `try_downloading=1`, file is downloaded from the G'MIC server and stored in the `${-path_cache}` folder.

## Default values:

`try_downloading_from_gmic_server=1`.

# input_csv

## Arguments:

- `"filename",_read_data_as={ 0:numbers | 1:strings | _variable_name }`

## Description:

Insert number of string array from specified .csv file.

If `variable_name` is provided, the string of each cell is stored in a numbered variable `_variable_name_x_y`, where `x` and `y` are the indices of the cell column and row respectively (starting from `0`).
Otherwise, a `WxH` image is inserted at the end of the list, with each vector-valued pixel `I(x,y)` encoding the number or the string of each cell.
This command returns the `W,H` dimension of the read array, as the status.

## Default values:

`read_data_as=1`.

# input_cube

## Arguments:

- `"filename",_convert_1d_cluts_to_3d={ 0 | 1 }.`

## Description:

Insert CLUT data from a .cube filename (Adobe CLUT file format).

## Default values:

`convert_1d_cluts_to_3d=1`.

---

# input_flo

## Arguments:

- `"filename"`

## Description:

Insert optical flow data from a .flo filename (vision.middlebury.edu file format).

---

# input_glob

## Arguments:

- `pattern`

## Description:

Insert new images from several filenames that match the specified glob pattern.

(*equivalent to shortcut command* `ig`).

---

# input_gpl

## Arguments:

- `filename`

## Description:

Input specified filename as a .gpl palette data file.

---

# input_obj

## Arguments:

- `filename`

## Description:

Input specified 3D mesh from a .obj Wavefront file.

---

# input_text

## Arguments:

- `filename`

## Description:

Input specified text-data filename as a new image.

(*equivalent to shortcut command* `it`).

---

# inrange

## Arguments:

- `min[%],max[%],_include_min_boundary={ 0:no | 1:yes },_include_max_boundary={ 0:no | 1:yes }`

## Description:

Detect pixels whose values are in specified range `[min,max]`, in selected images.

(*equivalent to shortcut command* `ir`).

## Default values:

`include_min_boundary=include_max_boundary=1`.

## Example of use:

```
image.jpg +inrange 25%,75%
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

# int2rgb

**No arguments**

## Description:

Convert color representation of selected images from INT24 to RGB.

# invert

Built-in command

## Arguments:

- `_use_LU={ 0:SVD | 1:LU },_lambda>=0`

## Description:

Inverse selected matrices (or compute Moore-Penrose pseudoinverse for non-square matrices).

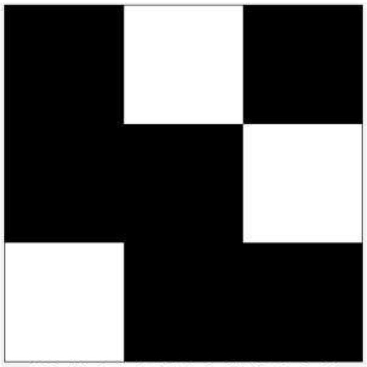
SVD solver is slower but more precise than LU.
`lambda` is used only in the Moore-Penrose pseudoinverse, by estimating A^t.(A^t.A + lambda.Id)^-1.

## Default values:

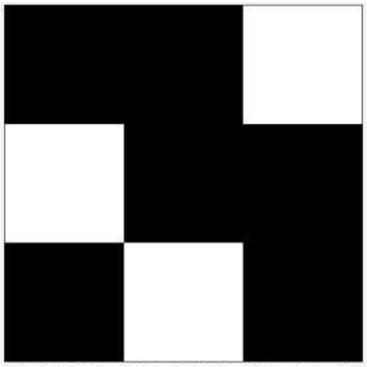
`use_LU=0` and `lambda=0`.

## Example of use:

```
(0,1,0;0,0,1;1,0,0) +invert
```



[0]: '(0,1,0;0,0,1;1,0,0)' (3x3x1x1)



[1]: '(0,1,0;0,0,1;1,0,0)_c1' (3x3x1x1)

---

# ipremula

**No arguments**

## Description:

Convert selected images with premultiplied alpha colors to normal colors.

## See also:

**premula**.

---

# is_change

## Arguments:

- `_value={ 0:false | 1:true }`

## Description:

Set or unset the `is_change` flag associated to the image list.

This flag tells the interpreter whether or not the image list should be displayed when the pipeline ends.

## Default values:

`value=1`.

---

# is_ext

## Arguments:

- `filename,_extension`

### Description:

Return 1 if specified filename has a given extension.

---

## is_half

#### No arguments

### Description:

Return 1 if the type of image pixels is limited to half-float.

---

## is_image_arg

### Arguments:

- `string`

### Description:

Return 1 if specified string looks like `[ind]`.

---

## is_macos

#### No arguments

### Description:

Return 1 if current computer OS is Darwin (MacOS), 0 otherwise.

---

## is_mesh3d

#### No arguments

### Description:

Return 1 if all of the selected images are 3D meshes, 0 otherwise.

---

## is_pattern

### Arguments:

- `string`

### Description:

Return 1 if specified string looks like a drawing pattern `0x.......`.

---

## is_varname

### Arguments:

- `string`

### Description:

Return 1 if specified string can be considered as a valid variable name.

---

## is_videofilename

### Arguments:

- `filename`

### Description:

Return 1 if extension of specified filename is typical from video files.

---

## is_windows

**No arguments**

### Description:

Return 1 if current computer OS is Windows, 0 otherwise.

---

## isoline3d

**Built-in command**

### Arguments:

- `isovalue[%]` or
- `'formula',value,_x0,_y0,_x1,_y1,_size_x>0[%],_size_y>0[%]`

### Description:

Extract 3D isolines with specified value from selected images or from specified formula.

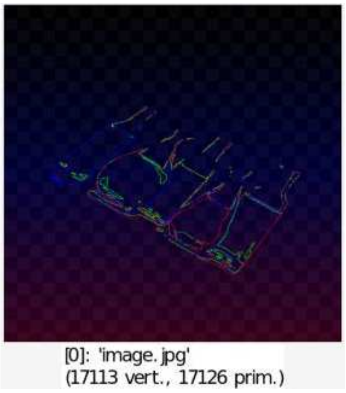
## Default values:

`x0=y0=-3`, `x1=y1=3` and `size_x=size_y=256`.

## Examples of use:

### • Example #1

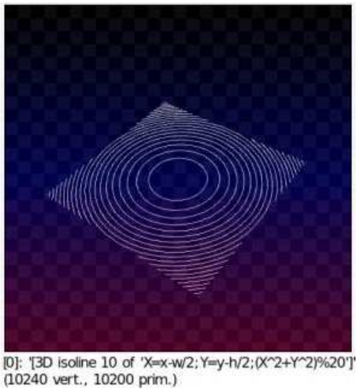
```
image.jpg blur 1 isoline3d 50%
```



[0]: 'image.jpg'
(17113 vert., 17126 prim.)

### • Example #2

```
isoline3d 'X=x-w/2;Y=y-h/2;(X^2+Y^2)%20',10,-10,-10,10,10
```



[0]: '[3D isoline 10 of 'X=x-w/2;Y=y-h/2;(X^2+Y^2)%20']'
(10240 vert., 10200 prim.)

---

# isophotes

## Arguments:

- `_nb_levels>0`

## Description:

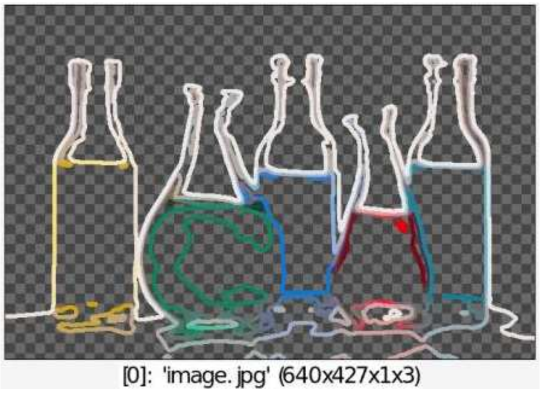
Render isophotes of selected images on a transparent background.

## Default values:

`nb_levels=64`

## Example of use:

```
image.jpg blur 2 isophotes 6 dilate_circ 5 display_rgba
```



---

# isosurface3d

**Built-in command**

## Arguments:

- `isovalue[%]` or
- `'formula',value,_x0,_y0,_z0,_x1,_y1,_z1,_size_x>0[%],_size_y>0[%],_size_z>0[`

## Description:

Extract 3D isosurfaces with specified value from selected images or from specified formula.

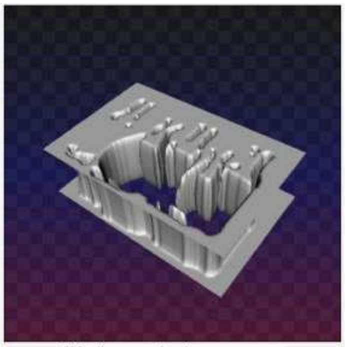
## Default values:

`x0=y0=z0=-3`, `x1=y1=z1=3` and `size_x=size_y=size_z=32`.

## Examples of use:

### • Example #1

```
image.jpg rescale2d ,128 luminance threshold 50% expand_z 2,0 blur 1
isosurface3d 50% mul3d 1,1,30
```



[0]: 'image.jpg'
(37252 vert., 73272 prim.)

- **Example #2**

```
isosurface3d 'x^2+y^2+abs(z)^abs(4*cos(x*y*z*3))',3
```

[0]: '[3D isosurface 3 of 'x^2+y^2+abs(z)^abs(4*cos(x*y*z*3))']'
(2928 vert., 5564 prim.)

---

# jzazbz2rgb

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from RGB to Jzazbz.

## Default values:

`illuminant=2`.

---

# jzazbz2xyz

**No arguments**

## Description:

Convert color representation of selected images from RGB to XYZ.

# kaleidoscope

## Arguments:

- `_center_x[%],_center_y[%],_radius,_angle,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

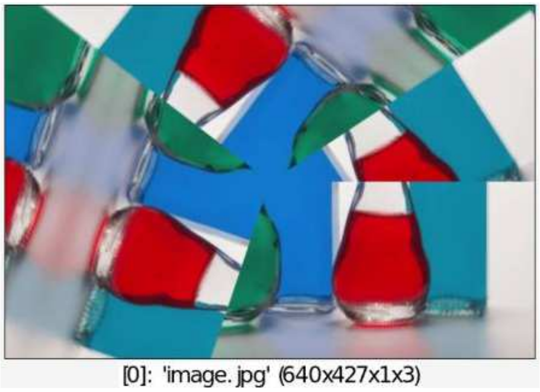
Create kaleidoscope effect from selected images.

## Default values:

`center_x=center_y=50%`, `radius=100`, `angle=30` and `boundary_conditions=3`.

## Example of use:

```
image.jpg kaleidoscope ,
```



# keep

Built-in command

### No arguments

## Description:

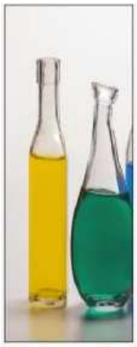
Keep only selected images.

(*equivalent to shortcut command* `k`).

## Examples of use:

**• Example #1**

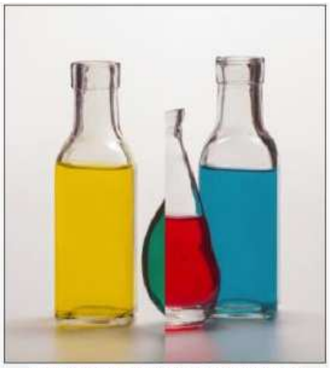
```
image.jpg split x keep[0-50%:2] append x
```



[0]: 'image.jpg'
(161x427x1x3)

- **Example #2**

```
image.jpg split x keep[^30%-70%] append x
```



[0]: 'image.jpg' (384x427x1x3)

# keep_named

## Arguments:

- `"name1","name2",...`

## Description:

Keep all images with specified names from the list of images.

Remove all images if no images with those names exist.

(*equivalent to shortcut command* `kmn`).

# kuwahara

### Arguments:

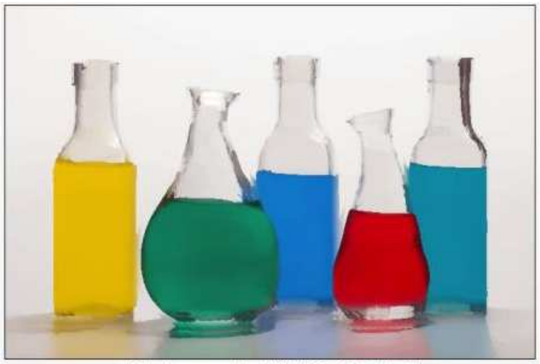
- `size>0`

### Description:

Apply Kuwahara filter of specified size on selected images.

### Example of use:

```
image.jpg kuwahara 9
```



[0]: 'image.jpg' (640x427x1x3)

---

## laar

**No arguments**

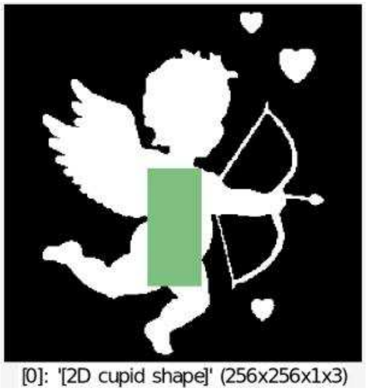
### Description:

Extract the largest axis-aligned rectangle in non-zero areas of selected images.

Rectangle coordinates are returned in status, as a sequence of numbers x0,y0,x1,y1.

### Example of use:

```
shape_cupid 256 coords=${-laar} normalize 0,255 to_rgb rectangle
$coords,0.5,0,128,0
```



[0]: '[2D cupid shape]' (256x256x1x3)

## lab2lch

**No arguments**

### Description:

Convert color representation of selected images from Lab to Lch.

## lab2rgb

### Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

### Description:

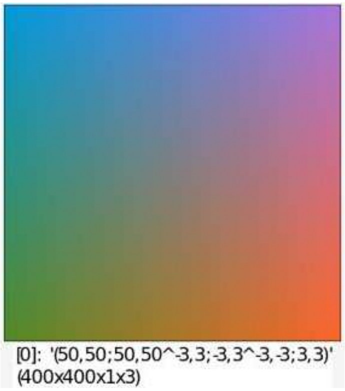
Convert color representation of selected images from Lab to RGB.

### Default values:

`illuminant=2`.

### Example of use:

```
(50,50;50,50^-3,3;-3,3^-3,-3;3,3) resize 400,400,1,3,3 lab2rgb
```



```
[0]: '(50,50;50,50^-3,3;-3,3^-3,-3;3,3)'
(400x400x1x3)
```

# lab2srgb

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

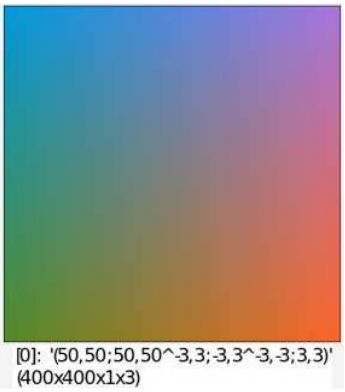
Convert color representation of selected images from Lab to sRGB.

## Default values:

`illuminant=2`.

## Example of use:

```
(50,50;50,50^-3,3;-3,3^-3,-3;3,3) resize 400,400,1,3,3 lab2rgb
```



```
[0]: '(50,50;50,50^-3,3;-3,3^-3,-3;3,3)'
(400x400x1x3)
```

# lab2xyz

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or

- `(no arg)`

## Description:

Convert color representation of selected images from Lab to XYZ.

## Default values:

`illuminant=2`.

---

# lab82rgb

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from Lab8 to RGB.

## Default values:

`illuminant=2`.

---

# lab82srgb

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

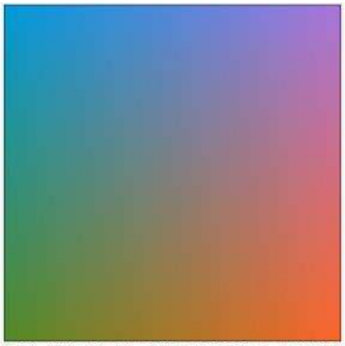
Convert color representation of selected images from Lab8 to sRGB.

## Default values:

`illuminant=2`.

## Example of use:

```
(50,50;50,50^-3,3;-3,3^-3,-3;3,3) resize 400,400,1,3,3 lab2rgb
```



[0]: '(50,50;50,50^-3,3;-3,3^-3,-3;3,3)' (400x400x1x3)

# label

Built-in command

## Arguments:

- `_tolerance>=0,is_high_connectivity={ 0 | 1 },_is_L2_norm={ 0 | 1 }`

## Description:

Label connected components in selected images.

If `is_L2_norm=1`, tolerances are compared against L2-norm, otherwise L1-norm is used.

## Default values:

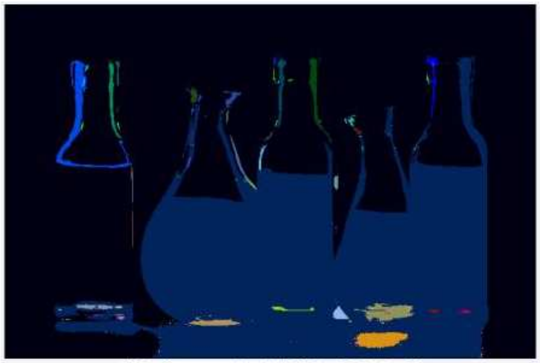
`tolerance=0`, `is_high_connectivity=0` and `is_L2_norm=1`.

This command has a **tutorial page**.

## Examples of use:

### • Example #1

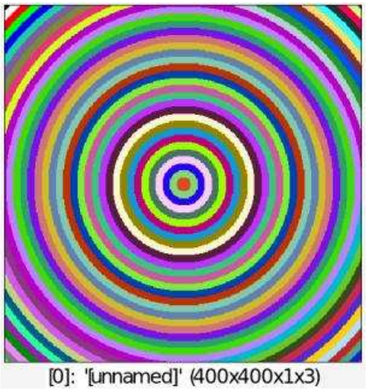
```
image.jpg luminance threshold 60% label normalize 0,255 map 0
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
400,400 set 1,50%,50% distance 1 mod 16 threshold 8 label mod 255 map
2
```


[0]: '[unnamed]' (400x400x1x3)

## label3d

### Arguments:

- `"text",font_height>=0,_opacity,_color1,...`

### Description:

Generate 3D text label.

### Default values:

`font_height=13`, `opacity=1` and `color=255,255,255`.

## label_fg

### Arguments:

- `tolerance>=0,is_high_connectivity={ 0 | 1 },_is_L2_norm={ 0 | 1 }`

### Description:

Label connected components for non-zero values (foreground) in selected images.

Similar to `label` except that 0-valued pixels are not labeled.
If `is_L2_norm=1`, tolerances are compared against L2-norm, otherwise L1-norm is used.

### Default values:

`is_high_connectivity=0`.

# label_points3d

## Arguments:

- `_label_size>0,_opacity`

## Description:

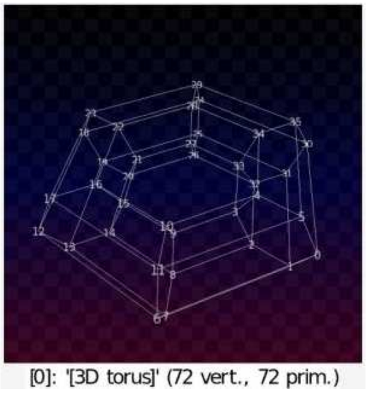
Add a numbered label to all vertices of selected 3D objects.

## Default values:

`label_size=13` and `opacity=0.8`.

## Example of use:

```
torus3d 100,40,6,6 label_points3d 23,1 mode3d 1
```



[0]: '[3D torus]' (72 vert., 72 prim.)

---

# laplacian

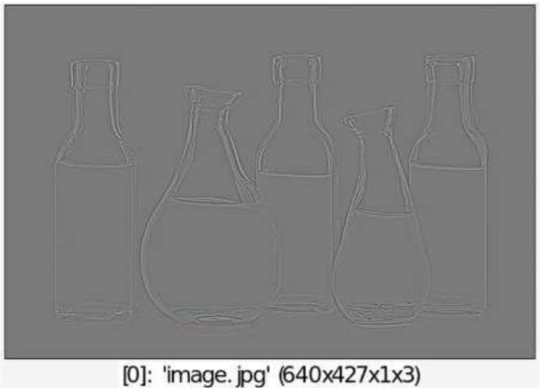
**No arguments**

## Description:

Compute Laplacian of selected images.

## Example of use:

```
image.jpg laplacian
```



[0]: 'image.jpg' (640x427x1x3)

---

# lathe3d

## Arguments:

- `_resolution>0,_smoothness[%]>=0,_max_angle>=0`

## Description:

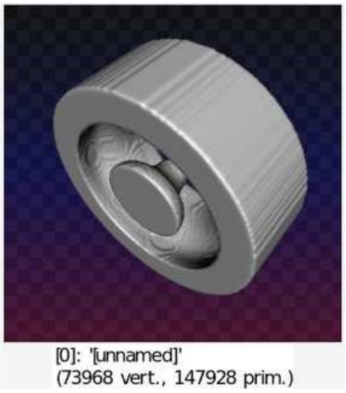
Generate 3D object from selected binary XY-profiles.

## Default values:

`resolution=128`, `smoothness=0.5%` and `max_angle=361`.

## Example of use:

```
300,300 rand -1,1 blur 40 sign normalize 0,255 lathe3d ,
```



[0]: '[unnamed]'
(73968 vert., 147928 prim.)

---

# lch2lab

**No arguments**

## Description:

Convert color representation of selected images from Lch to Lab.

# lch2rgb

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from Lch to RGB.

## Default values:

`illuminant=2`.

# lch82rgb

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from Lch8 to RGB.

## Default values:

`illuminant=2`.

# le

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

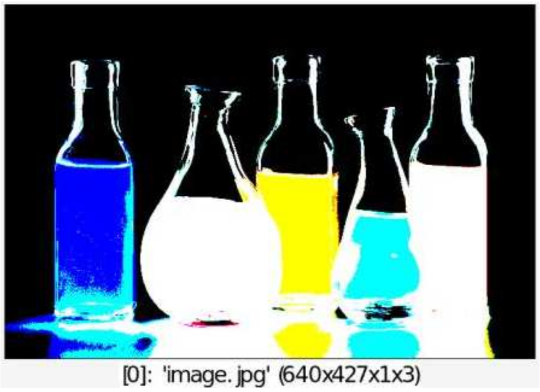
Compute the boolean 'less or equal than' of selected images with specified value, image or mathematical expression, or compute the boolean 'less or equal than' of selected images.

(*equivalent to shortcut command* `<=`).

## Examples of use:

### • Example #1

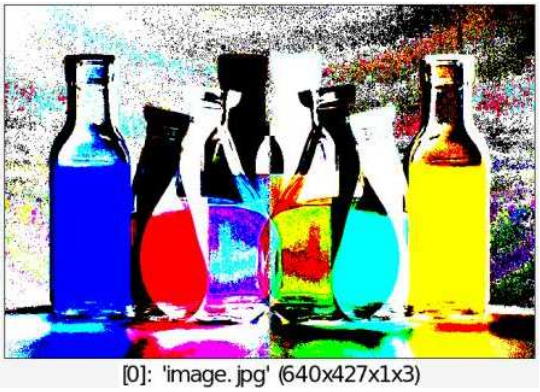
```
image.jpg le {ia}
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +mirror x le
```



[0]: 'image.jpg' (640x427x1x3)

---

# lic

## Arguments:

- `_amplitude>0,_channels>0`

## Description:

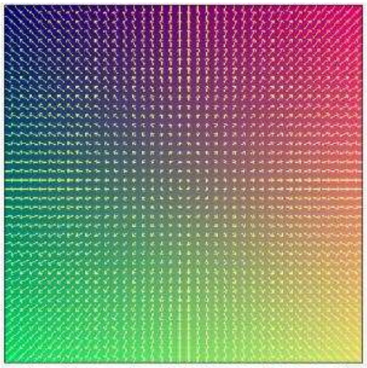
Render LIC representation of selected vector fields.

## Default values:

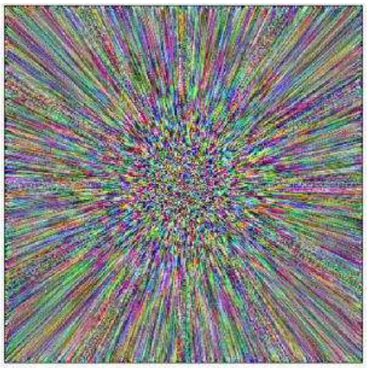
`amplitude=30` and `channels=1`.

## Example of use:

```
400,400,1,2,'!c?x-w/2:y-h/2' +lic 200,3 quiver[-2] [-2],10,1,1,1,255
```



[0]: '[!c?x-w/2:y-h/2]' (400x400x1x2)



[1]: '[!c?x-w/2:y-h/2]_c1' (400x400x1x3)

# light3d

Built-in command

## Arguments:

- `position_x,position_y,position_z` or
- `[texture]` or
- `(no arg)`

## Description:

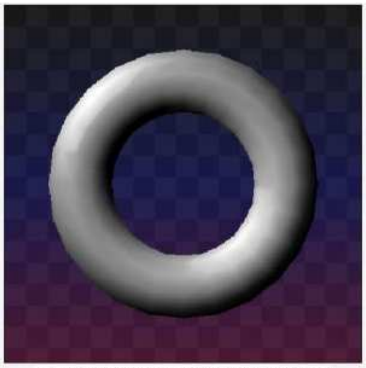
Set the light coordinates or the light texture for 3D rendering.

(*equivalent to shortcut command* `l3d`).

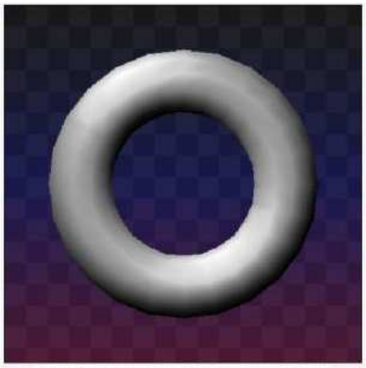
(no arg) resets the 3D light to default.

## Example of use:

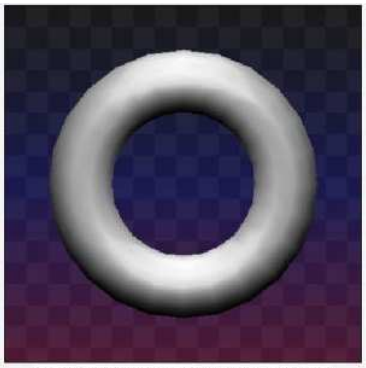
```
torus3d 100,30 double3d 0 specs3d 1.2 repeat 5 { light3d {$>*100},0,
-300 +snapshot3d[0] 400 } remove[0]
```



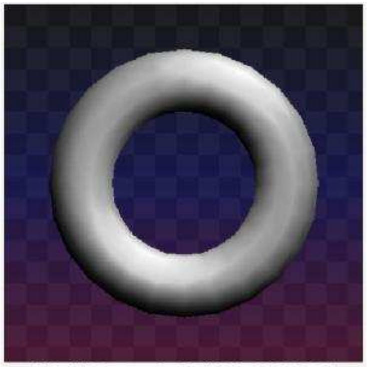
[0]: '[3D torus]_c1' (400x400x1x3)



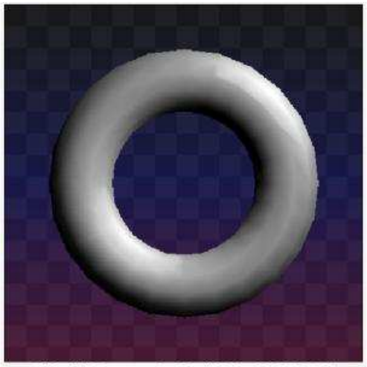
[1]: '[3D torus]_c1' (400x400x1x3)



[2]: '[3D torus]_c1' (400x400x1x3)



[3]: '[3D torus]_c1' (400x400x1x3)



[4]: '[3D torus]_c1' (400x400x1x3)

# light_patch

## Arguments:

- `_density>0,_darkness>=0,_lightness>=0`

## Description:

Add light patches to selected images.

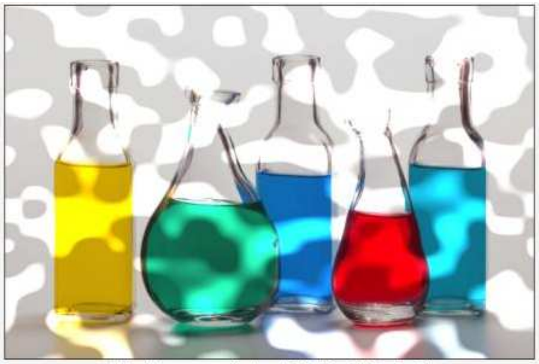
## Default values:

`density=10`, `darkness=0.9` and `lightness=1.7`.

## Example of use:

```
image.jpg +light_patch 20,0.9,4
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# light_relief

## Arguments:

- `_ambient_light,_specular_lightness,_specular_size,_darkness,_light_smoothness`
  `0 | 1 }`

### Description:

Apply relief light to selected images.

Default values(s) : `ambient_light=0.3`, `specular_lightness=0.5`, `specular_size=0.2`, `darkness=0`, `xl=0.2`, `yl=zl=0.5`,
`zscale=1`, `opacity=1` and `opacity_is_heightmap=0`.

### Example of use:

```
image.jpg blur 2 light_relief 0.3,4,0.1,0
```

[0]: 'image.jpg' (640x427x1x3)

## lightness

**No arguments**

### Description:

Compute lightness of selected sRGB images.

### Example of use:

```
image.jpg +lightness
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x1)

# lightrays

## Arguments:

- `100<=_density<=0,_center_x[%],_center_y[%],_ray_length>=0,_ray_attenuation>=0`

## Description:

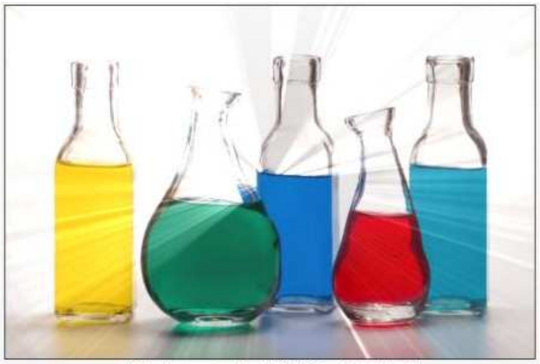
Generate ray lights from the edges of selected images.

## Default values:

`density=50%`, `center_x=50%`, `center_y=50%`, `ray_length=0.9` and `ray_attenuation=0.5`.

## Example of use:

```
image.jpg +lightrays , + cut 0,255
```



[0]: 'image.jpg' (640x427x1x3)

# line

Built-in command

## Arguments:

- `x0[%],y0[%],x1[%],y1[%],_opacity,_pattern,_color1,...`

## Description:

Draw specified colored line on selected images.

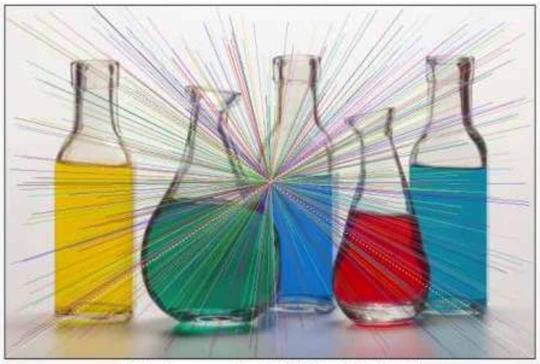
`pattern` is an hexadecimal number starting with `0x` which can be omitted even if a color is specified.

## Default values:

`opacity=1`, `pattern=(undefined)` and `color1=0`.

## Example of use:

```
image.jpg repeat 500 line 50%,50%,{u(w)},{u(h)},0.5,${-rgb} done line
0,0,100%,100%,1,0xCCCCCCCC,255 line 100%,0,0,100%,1,0xCCCCCCCC,255
```



[0]: 'image.jpg' (640x427x1x3)

## line3d

### Arguments:

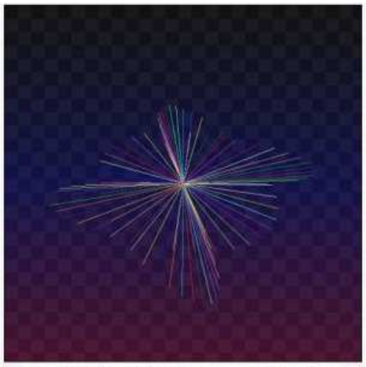
- `x0,y0,z0,x1,y1,z1`

### Description:

Input 3D line at specified coordinates.

### Example of use:

```
repeat 100 { a:=$>*pi/50 line3d 0,0,0,{cos(3*$a)},{sin(2*$a)},0
color3d. ${-rgb} } add3d
```



[0]: '[3D line]' (200 vert., 100 prim.)

## line_aa

### Arguments:

- `x0[%],y0[%],x1[%],y1[%],_opacity,_color1,...`

## Description:

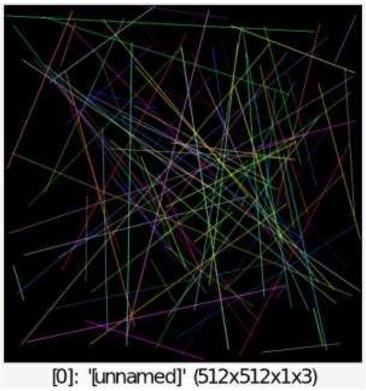
Draw specified antialiased colored line on selected images.

## Default values:

`opacity=1` and `color1=0`.

## Example of use:

```
512,512,1,3 repeat 100 line_aa {v([w,h,w,h])-1},1,${-RGB} done
```



# linearize_tiles

## Arguments:

- `M>0,_N>0`

## Description:

Linearize MxN tiles on selected images.

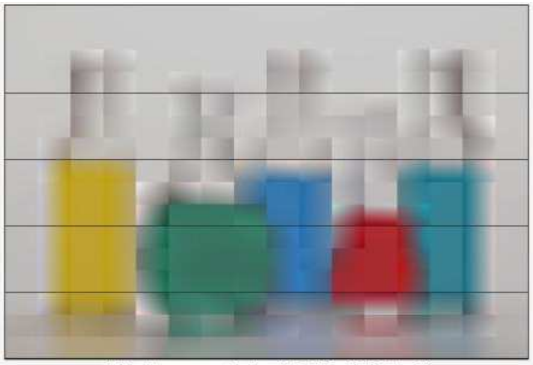
## Default values:

`N=M`.

## Example of use:

```
image.jpg +linearize_tiles 16
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x432x1x3)

# lines3d

## Arguments:

- `_length>=0`

## Description:

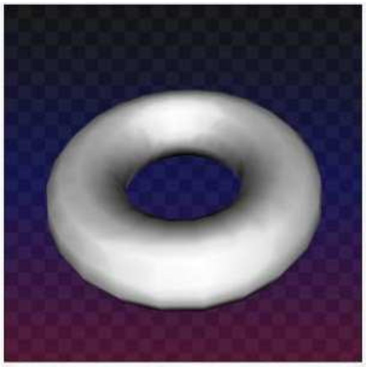
Convert specified 3D objects to sets of 3D horizontal segments with specified length.

## Default values:

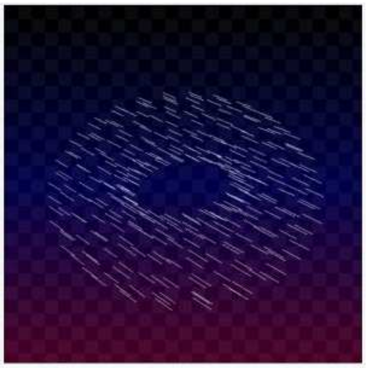
`length=1`.

## Example of use:

```
torus3d 100,40 +lines3d 20
```



[0]: '[3D torus]' (288 vert., 288 prim.)



[1]: '[3D torus]_c1' (576 vert., 288 prim.)

# linify

## Arguments:

- `0<=_density<=100,_spreading>=0,_resolution[%]>0,_line_opacity>=0,_line_precis`
  `0:subtractive | 1:additive }`

## Description:

Apply linify effect on selected images.

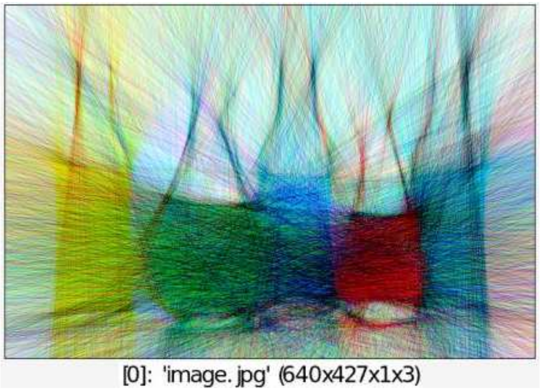
The algorithm is inspired from the one described on the webpage **http://linify.me/about**.

## Default values:

`density=50`, `spreading=2`, `resolution=40%`, `line_opacity=10`, `line_precision=24` and `mode=0`.

## Example of use:

```
image.jpg linify 60
```



[0]: 'image.jpg' (640x427x1x3)

# lissajous3d

## Arguments:

- `resolution>1,a,A,b,B,c,C`

## Description:

Input 3D lissajous curves `x(t)=sin(a*t+A*2*pi)`, `y(t)=sin(b*t+B*2*pi)`, `z(t)=sin(c*t+C*2*pi)`.

## Default values:

`resolution=1024`, `a=2`, `A=0`, `b=1`, `B=0`, `c=0` and `C=0`.

## Example of use:

```
lissajous3d ,
```

[0]: '[3D lissajou]'
(1024 vert., 1023 prim.)

---

# local

Built-in command

### No arguments

## Description:

Start a `local...[onfail]...done` block, with selected images.

(*equivalent to shortcut command* `l`).

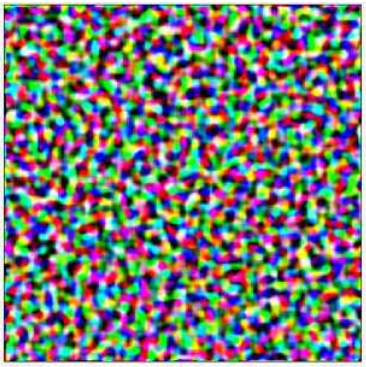
This command has a **tutorial page**.

## Examples of use:

### • Example #1

```
image.jpg local[] 300,300,1,3 rand[0] 0,255 blur 4 sharpen 1000 done
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[unnamed]' (300x300x1x3)

### • Example #2

```
image.jpg +local repeat 3 { deform 20 } done
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

# lof

## Arguments:

- `feature`

## Description:

Return the list of specified features (separated by commas) for each selected images.

# log

Built-in command

**No arguments**

## Description:

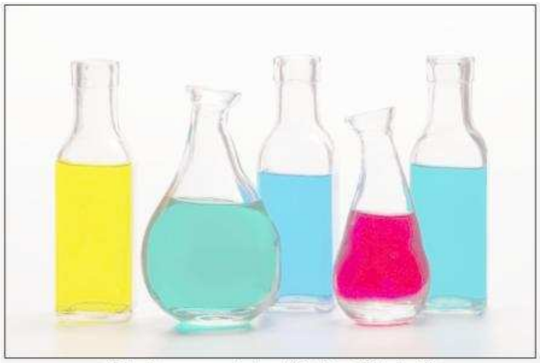
Compute the pointwise base-e logarithm of selected images.

## Examples of use:

**• Example #1**

```
image.jpg +add 1 log[-1]
```

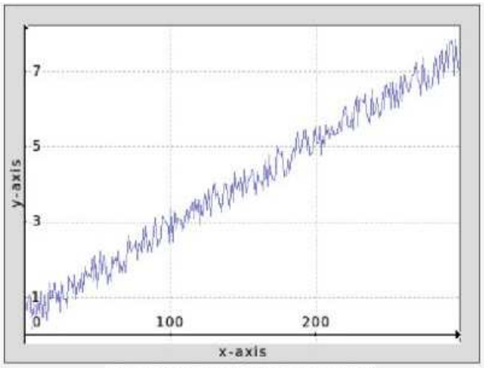
[0]: 'image.jpg' (640x427x1x3)



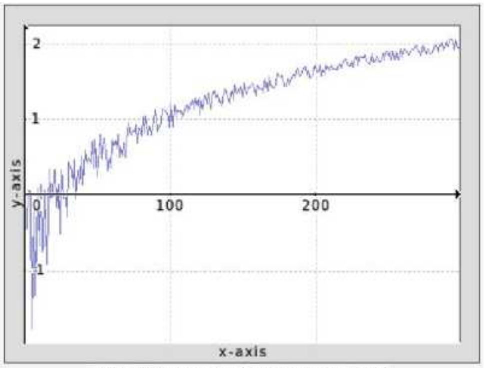
[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
300,1,1,1,'7*x/w+u' +log display_graph 400,300
```



[0]: '[7*x/w+u]' (400x300x1x3)



[1]: '[7*x/w+u]_c1' (400x300x1x3)

# log10

Built-in command

**No arguments**

## Description:

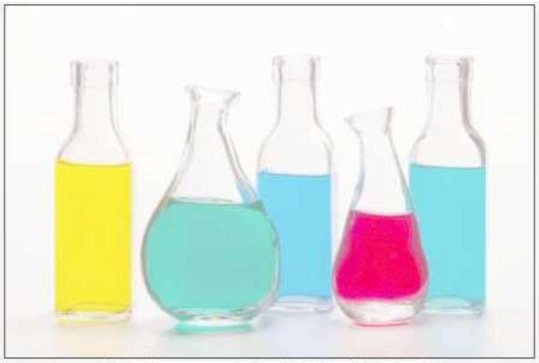
Compute the pointwise base-10 logarithm of selected images.

## Examples of use:

- **Example #1**

```
image.jpg +add 1 log10[-1]
```

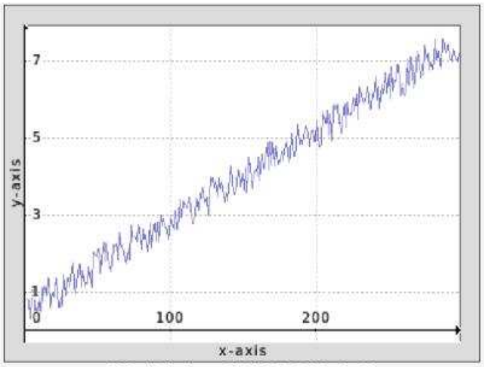
[0]: 'image.jpg' (640x427x1x3)



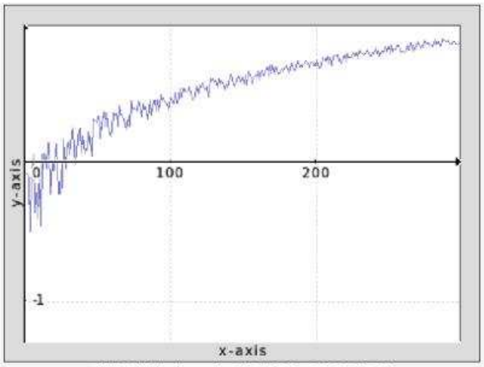
[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
300,1,1,1,'7*x/w+u' +log10 display_graph 400,300
```



[0]: '[7*x/w+u]' (400x300x1x3)



[1]: '[7*x/w+u]_c1' (400x300x1x3)

# log2

Built-in command

**No arguments**

## Description:

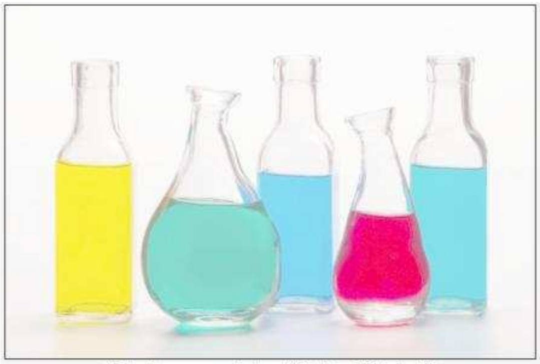
Compute the pointwise base-2 logarithm of selected images

## Examples of use:

### • Example #1

```
image.jpg +add 1 log2[-1]
```

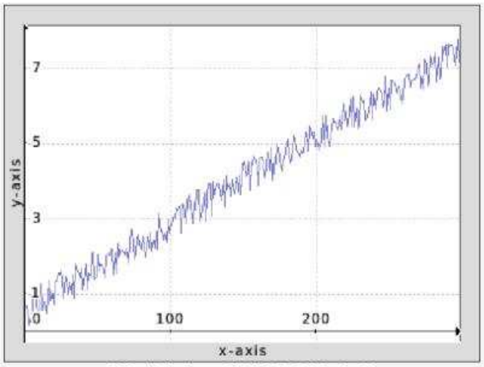
[0]: 'image.jpg' (640x427x1x3)



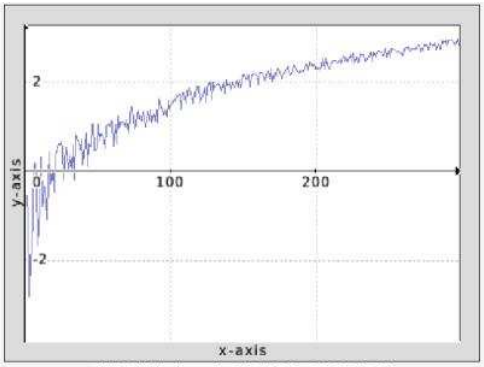
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'7*x/w+u' +log2 display_graph 400,300
```



[0]: '[7*x/w+u]' (400x300x1x3)



[1]: '[7*x/w+u]_c1' (400x300x1x3)

# lorem

## Arguments:

- `_width>0,_height>0`

## Description:

Input random image of specified size, retrieved from Internet.

## Default values:

`width=height=800`.

# lt

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

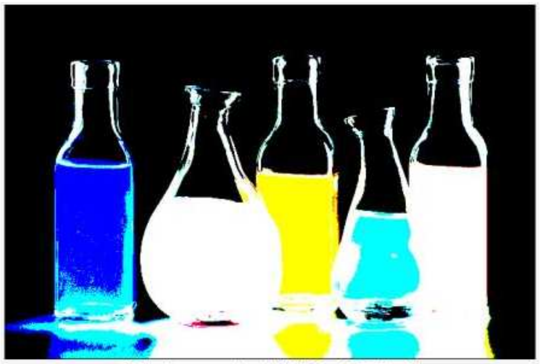
Compute the boolean 'less than' of selected images with specified value, image or mathematical expression, or compute the boolean 'less than' of selected images.

(*equivalent to shortcut command* `<`).

## Examples of use:

**• Example #1**

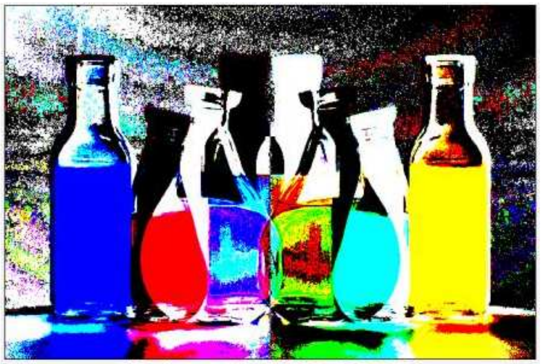
```
image.jpg lt {ia}
```



[0]: 'image.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg +mirror x lt
```



[0]: 'image.jpg' (640x427x1x3)

# luminance

### No arguments

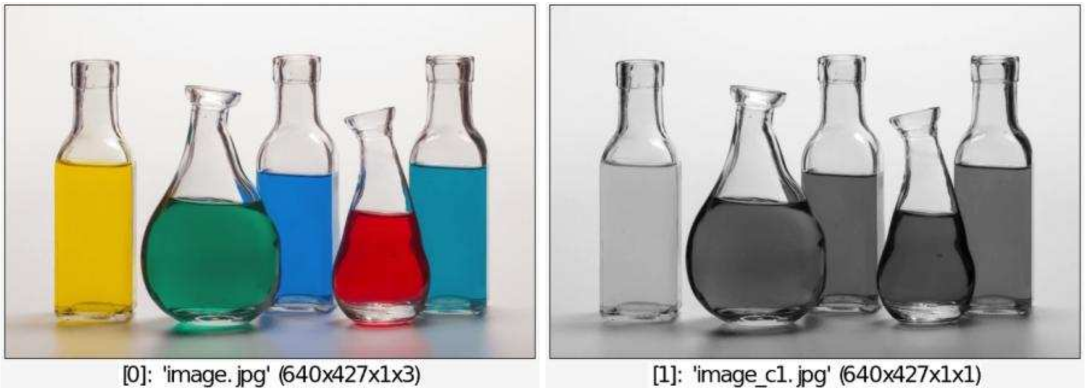
## Description:

Compute luminance of selected sRGB images.

This command has a **tutorial page**.

## Example of use:

```
image.jpg +luminance
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x1)

# lut_contrast

## Arguments:

- `_nb_colors>1,_min_rgb_value`

## Description:

Generate a RGB colormap where consecutive colors have high contrast.

This function performs a specific score maximization to generate the result, so it may take some time when `nb_colors` is high.

## Default values:

`nb_colors=256` and `min_rgb_value=64`.

# mad

**No arguments**

## Description:

Return the MAD (Maximum Absolute Deviation) of the last selected image.

The MAD is defined as MAD = med_i|x_i-med_j(x_j)|

# mandelbrot

Built-in command

## Arguments:

- `z0r,z0i,z1r,z1i,_iteration_max>=0,_is_julia={ 0 | 1 },_c0r,_c0i,_opacity`

## Description:

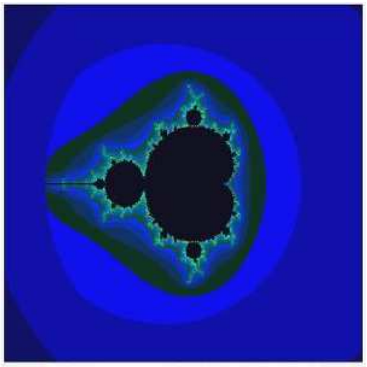
Draw mandelbrot/julia fractal on selected images.

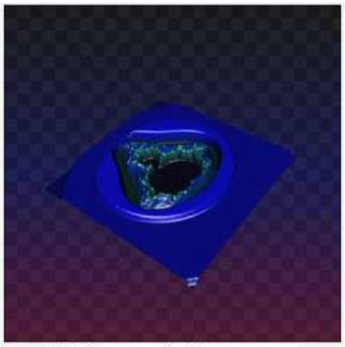
### Default values:

`iteration_max=100`, `is_julia=0`, `c0r=c0i=0` and `opacity=1`.

### Example of use:

```
400,400 mandelbrot -2.5,-2,2,2,1024 map 0 +blur 2 elevation3d[-1] -0.2
```


[0]: '[unnamed]' (400x400x1x3)


[1]: '[unnamed]_c1'
(160000 vert., 159201 prim.)

## map

**Built-in command**

### Arguments:

- `[palette],_boundary_conditions` or
- `palette_name,_boundary_conditions`

### Description:

Map specified vector-valued palette to selected indexed images.

Each output image has `M*N` channels, where `M` and `N` are the numbers of channels of, respectively, the corresponding input image and the `palette` image.
`palette_name` can be ***{ default | hsv | lines | hot | cool | jet | flag | cube | rainbow | algae | amp | balance | curl | deep | delta | dense | diff | gray | haline | ice | matter | oxy | phase | rain | solar | speed | tarn | tempo | thermal | topo | turbid | aurora | hocuspocus | srb2 | uzebox }***.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

### Default values:

`boundary_conditions=0`.

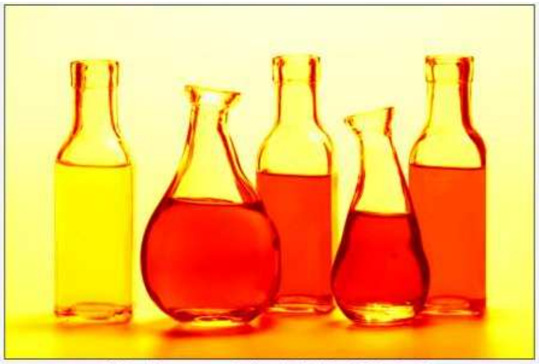
This command has a **tutorial page**.

### Examples of use:

- **Example #1**

```
image.jpg +luminance map[-1] 3
```

[0]: 'image.jpg' (640x427x1x3)

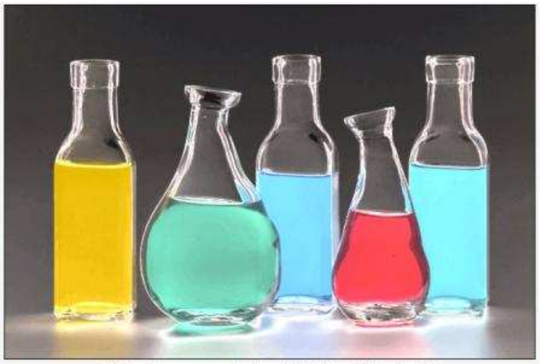


[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg +rgb2ycbcr split[-1] c (0,255,0) resize[-1] 256,1,1,1,3
map[-4] [-1] remove[-1] append[-3--1] c ycbcr2rgb[-1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# map_clut

## Arguments:

- `[clut] | "clut_name"`

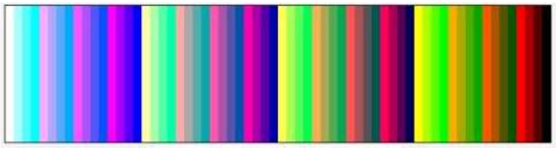
## Description:

Map specified RGB color LUT to selected images.

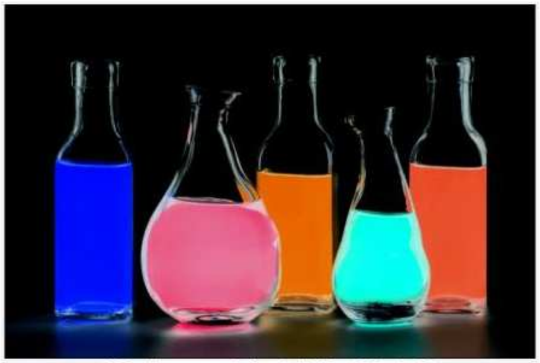
## Example of use:

```
image.jpg uniform_distribution {2^6},3 mirror[-1] x +map_clut[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[empty]' (64x1x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# map_sphere

## Arguments:

- `_width>0,_height>0,_radius,_dilation>0,_fading>=0,_fading_power>=0`

## Description:

Map selected images on a sphere.

## Default values:

`width=height=512`, `radius=100`, `dilation=0.5`, `fading=0` and `fading_power=0.5`.

## Example of use:

```
image.jpg map_sphere ,
```

[0]: 'image.jpg' (512x512x1x3)

# map_sprites

## Arguments:

- `_nb_sprites>=1,_allow_rotation={ 0:none | 1:90 deg. | 2:180 deg. }`

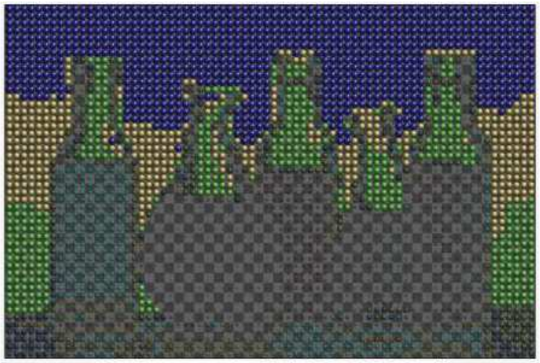
## Description:

Map set of sprites (defined as the `nb_sprites` latest images of the selection) to other selected images,

according to the luminosity of their pixel values.

## Example of use:

```
image.jpg rescale2d ,48 repeat 16 ball {8+2*$>},${-rgb} mul[-1] {(1+$>)/16} done map_sprites 16
```



[0]: '[unnamed]_c2' (2736x1824x1x4)

# map_tones

## Arguments:

- `_threshold>=0,_gamma>=0,_smoothness>=0,nb_iter>=0`

### Description:

Apply tone mapping operator on selected images, based on Poisson equation.

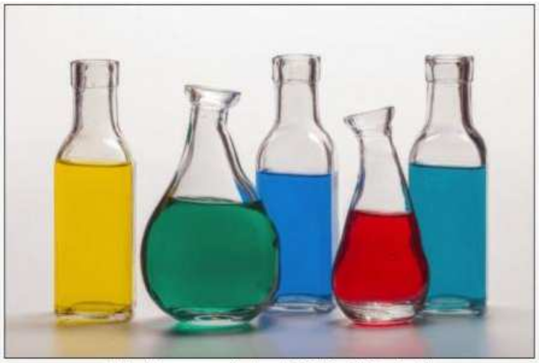
### Default values:

`threshold=0.1`, `gamma=0.8`, `smoothness=0.5` and `nb_iter=30`.

### Example of use:

```
image.jpg +map_tones ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## map_tones_fast

### Arguments:

- `_radius[%]>=0,_power>=0`

### Description:

Apply fast tone mapping operator on selected images.

### Default values:

`radius=3%` and `power=0.3`.

### Example of use:

```
image.jpg +map_tones_fast ,
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

# marble

## Arguments:

- `_image_weight,_pattern_weight,_angle,_amplitude,_sharpness>=0,_anisotropy>=0,`

## Description:

Render marble like pattern on selected images.

## Default values:

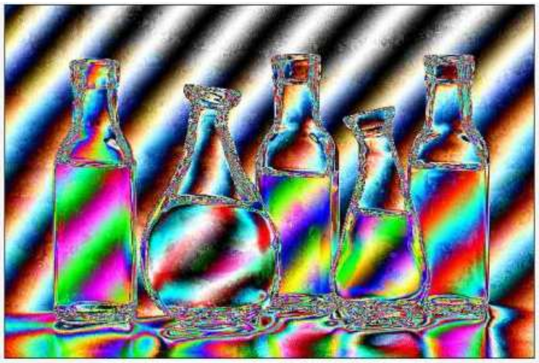
`image_weight=0.2`, `pattern_weight=0.1`, `angle=45`, `amplitude=0`, `sharpness=0.4` and `anisotropy=0.8`,

`alpha=0.6`, `sigma=1.1` and `cut_low=cut_high=0`.

## Example of use:

```
image.jpg +marble ,
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

# match_histogram

## Arguments:

- `[reference_image],_nb_levels>0,_color_channels`

## Description:

Transfer histogram of the specified reference image to selected images.

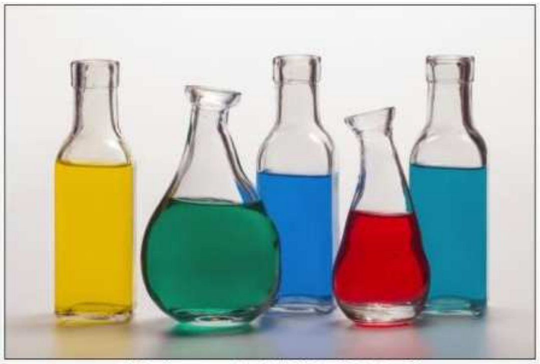
Argument 'color channels' is the same as with command `apply_channels`.

## Default values:

`nb_levels=256` and `color_channels=all`.

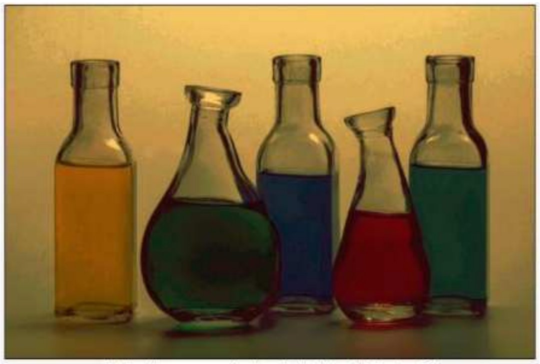
## Example of use:

```
image.jpg 100,100,1,3,"u([256,200,100])" +match_histogram[0] [1]
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'u([256,200,100])' (100x100x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

---

# match_icp

## Arguments:

- `[reference_image],_precision>0,_transformation_variable`

## Description:

Transform selected set of d-dimensional vectors to match specified set of reference vectors, using

ICP (*Iterative Closest Point*) algorithm.

A description of ICP is available at **https://en.wikipedia.org/wiki/Iterative_closest_point**.
Return the L2 alignment error.

## Default values:

`precision=1e-2` and `transformation_variable=(undefined)`.

sample lena,earth +match_icp[0] [1]

---

# match_pca

## Arguments:

- `[reference_image],_color_channels`

## Description:

Transfer mean and covariance matrix of specified vector-valued reference image to selected images.

Argument 'color channels' is the same as with command `apply_channels`.

## Default values:

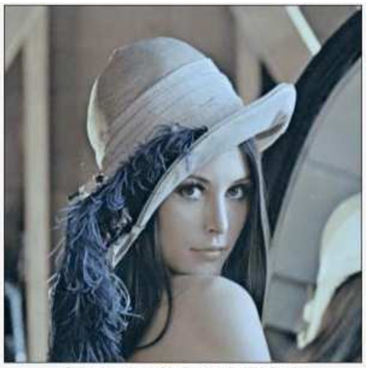
`color_channels=all`.

## Example of use:

```
sample lena,earth +match_pca[0] [1]
```

[0]: 'lena' (512x512x1x3)

[1]: 'earth' (500x500x1x3)



[2]: 'lena_c1' (512x512x1x3)

---

# match_rgb

## Arguments:

- `[target],_gamma>=0,_regularization>=0,_luminosity_constraints>=0,_rgb_resolut`

`0 | 1 }`

## Description:

Transfer colors from selected source images to selected reference image (given as argument).

`gamma` determines the importance of color occurrences in the matching process (0:none to 1:huge).
`regularization` determines the number of guided filter iterations to remove quantization effects.
`luminosity_constraints` tells if luminosity constraints must be applied on non-confident matched colors.
`is_constraints` tells if additional hard color constraints must be set (opens an interactive window).

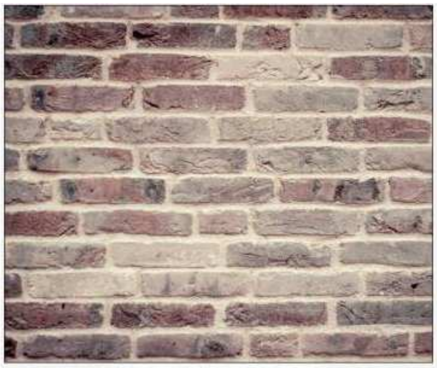
## Default values:

`gamma=0.3`, `regularization=8`, `luminosity_constraints=0.1`, `rgb_resolution=64` and `is_constraints=0`.

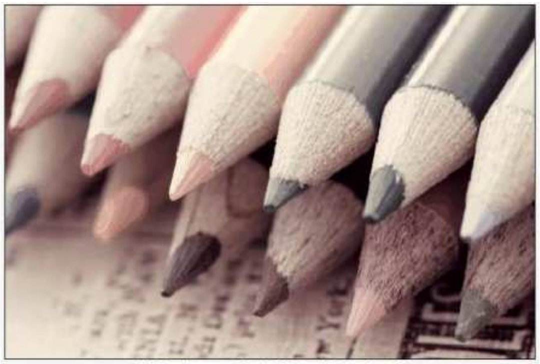
## Example of use:

```
sample pencils,wall +match_rgb[0] [1],0,0.01
```

[0]: 'pencils' (600x400x1x3)



[1]: 'wall' (600x500x1x3)



[2]: 'res_c1' (600x400x1x3)

# matchpatch

**Built-in command**

## Arguments:

- `[patch_image],patch_width>=1,_patch_height>=1,_patch_depth>=1,_nb_iterations>`
  `0 | 1 },_[guide]`

## Description:

Estimate correspondence map between selected images and specified patch image, using

a patch-matching algorithm.
Each pixel of the returned correspondence map gives the location (p,q) of the closest patch in the specified patch image. If `output_score=1`, the third channel also gives the corresponding matching score for each patch as well.
If `patch_penalization` is >=0, SSD is penalized with patch occurrences.
If `patch_penalization` is <0, SSD is inf-penalized when distance between patches are less than `-patch_penalization`.

## Default values:

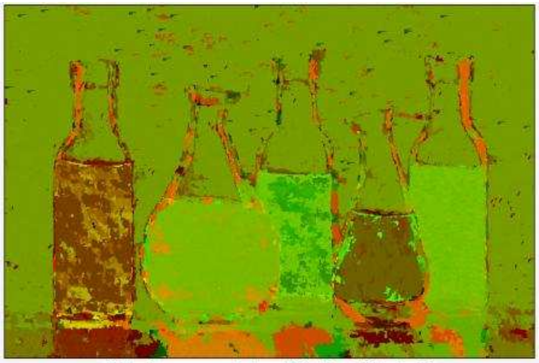
`patch_height=patch_width`, `patch_depth=1`, `nb_iterations=5`, `nb_randoms=5`, `patch_penalization=0`, `output_score=0` and `guide=(undefined)`.

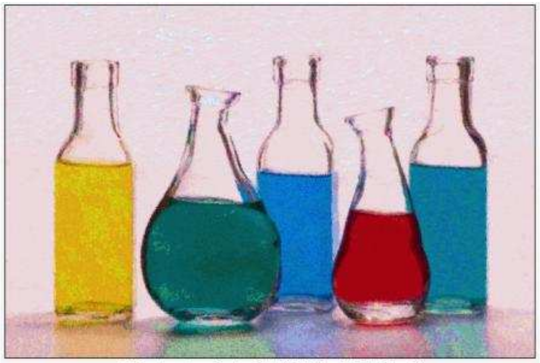
## Example of use:

```
image.jpg sample colorful +matchpatch[0] [1],3 +warp[-2] [-1],0
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'colorful' (800x800x1x3)


[2]: 'image_c1.jpg' (640x427x1x2)


[3]: 'colorful_c1' (640x427x1x3)

---

# math_lib

### No arguments

### Description:

Return string that defines a set of several useful macros for the embedded math evaluator.

---

## max

Built-in command

### Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

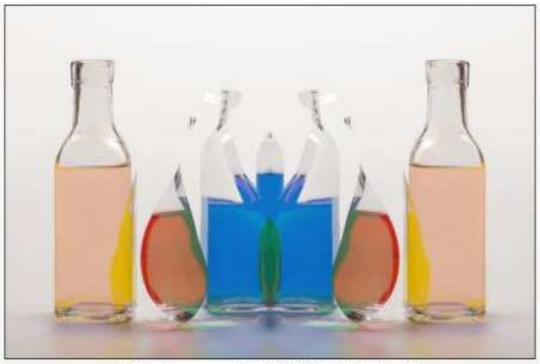
### Description:

Compute the maximum between selected images and specified value, image or mathematical expression, or compute the pointwise maxima between selected images.

### Examples of use:

**• Example #1**

```
image.jpg +mirror x max
```



[0]: 'image.jpg' (640x427x1x3)

**• Example #2**

```
image.jpg max 'R=((x/w-0.5)^2+(y/h-0.5)^2)^0.5;255*R'
```

[0]: 'image.jpg' (640x427x1x3)

## max_d

**No arguments**

### Description:

Return the maximal depth between selected images.

## max_h

**No arguments**

### Description:

Return the maximal height between selected images.

## max_patch

### Arguments:

- `_patch_size>=1`

### Description:

Return locations of maximal values in local patch-based neighborhood of given size for selected images.

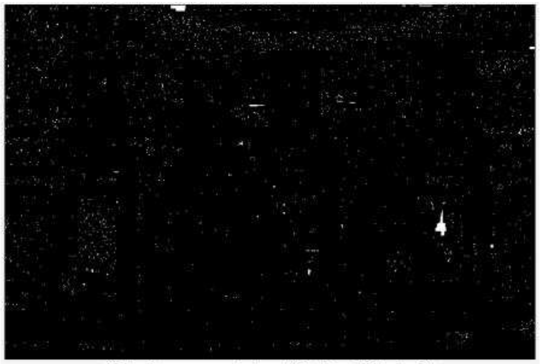
### Default values:

`patch_size=16`.

### Example of use:

```
image.jpg norm +max_patch 16
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)

## max_s

**No arguments**

### Description:

Return the maximal spectrum between selected images.

## max_w

**No arguments**

### Description:

Return the maximal width between selected images.

## max_wh

**No arguments**

### Description:

Return the maximal wxh size of selected images.

## max_whd

**No arguments**

### Description:

Return the maximal wxhxd size of selected images.

# max_whds

**No arguments**

## Description:

Return the maximal wxhxdxs size of selected images.

# maxabs

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

Compute the maxabs between selected images and specified value, image or mathematical expression, or compute the pointwise maxabs between selected images.

# maze

## Arguments:

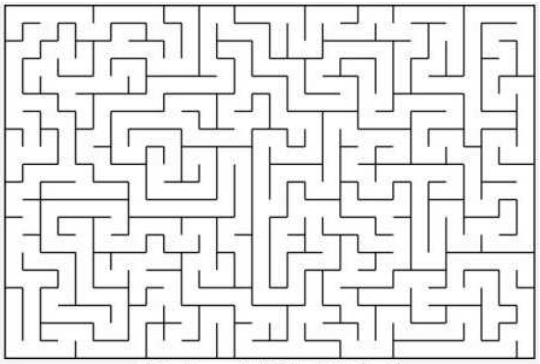
- `_width>0,_height>0,_cell_size>0`

## Description:

Input maze with specified size.

## Example of use:

```
maze 30,20 negate normalize 0,255
```



[0]: '[maze]' (720x480x1x1)

## maze_mask

### Arguments:

- `_cellsize>0`

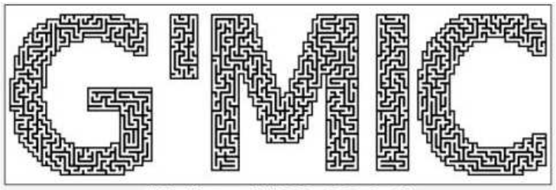
### Description:

Input maze according to size and shape of selected mask images.

Mask may contain disconnected shapes.

### Example of use:

```
0 text "G'MIC",0,0,53,1,1 dilate 3 autocrop 0 frame 1,1,0 maze_mask 8
dilate 3 negate mul 255
```



[0]: '[maze]' (952x312x1x1)

## mdiv

**Built-in command**

### Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

### Description:

Compute the matrix division of selected matrices/vectors by specified value, image or mathematical expression, or compute the matrix division of selected images.

(*equivalent to shortcut command* `m/`).

# meancurvature_flow

## Arguments:

- `_nb_iter>=0,_dt,_keep_sequence={ 0 | 1 }`

## Description:

Apply iterations of the mean curvature flow on selected images.

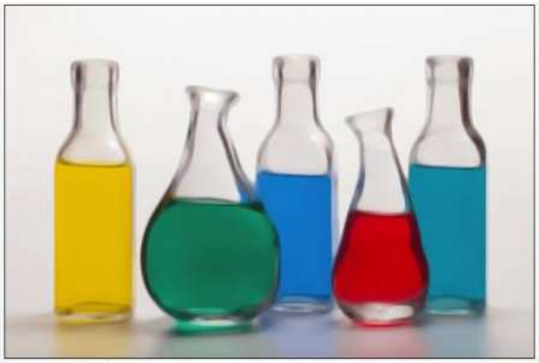
## Default values:

`nb_iter=10`, `dt=30` and `keep_sequence=0`.

## Example of use:

```
image.jpg +meancurvature_flow 20
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# med

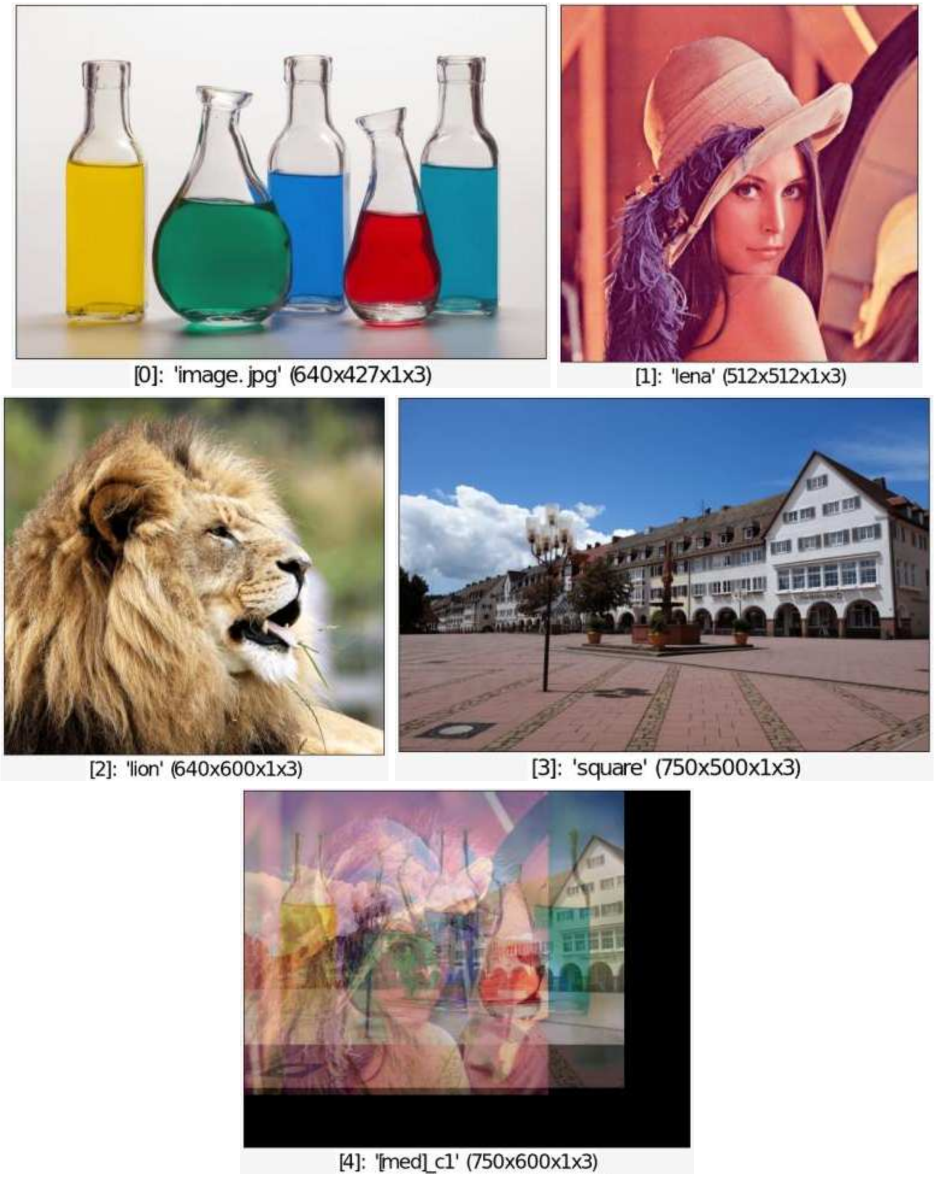
### No arguments

## Description:

Compute the median of selected images.

## Example of use:

```
image.jpg sample lena,lion,square +med
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'lena' (512x512x1x3)
[2]: 'lion' (640x600x1x3)
[3]: 'square' (750x500x1x3)
[4]: '[med]_c1' (750x600x1x3)

# median

Built-in command

## Arguments:

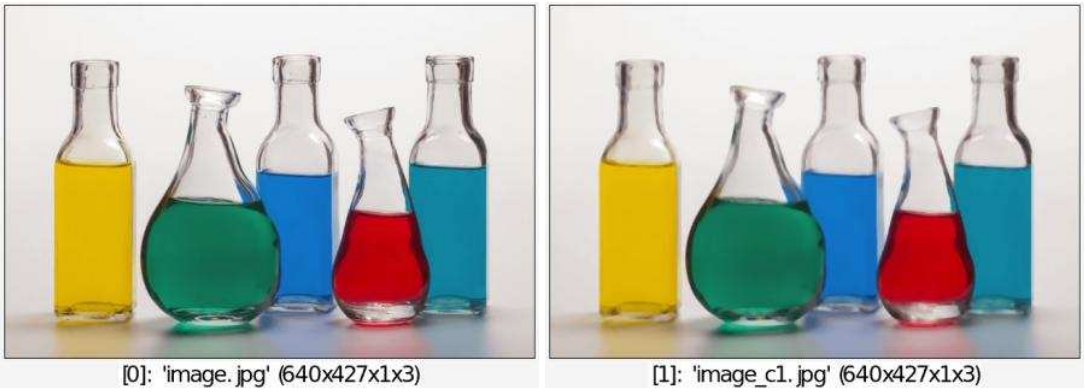
- `size>=0,_threshold>0`

## Description:

Apply (opt. thresholded) median filter on selected images with structuring element size x size.

## Example of use:

```
image.jpg +median 5
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

## median_files

### Arguments:

- `"filename_pattern",_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1,_frame_rows[%]>=1,_is_fast_approximation={ 0 | 1 }`

### Description:

Compute the median frame of specified input image files, in a streamed way.

If a display window is opened, rendered frame is displayed in it during processing.

### Default values:

`first_frame=0`, `last_frame=-1`, `frame_step=1`, `frame_rows=20%` and `is_fast_approximation=0`.

## median_vectors

**No arguments**

### Description:

Return the median vector value of the last selected image (median computed channel by channel)

## median_video

### Arguments:

- `video_filename,_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1,_frame_rows[%]>=1,_is_fast_approximation={ 0 | 1 }`

### Description:

Compute the median of all frames of an input video file, in a streamed way.

If a display window is opened, rendered frame is displayed in it during processing.
This command requires features from the OpenCV library (not enabled in G'MIC by default).

## Default values:

`first_frame=0`, `last_frame=-1`, `frame_step=1`, `frame_rows=100%` and `is_fast_approximation=1`.

---

# meigen

## Arguments:

- `m>=1`

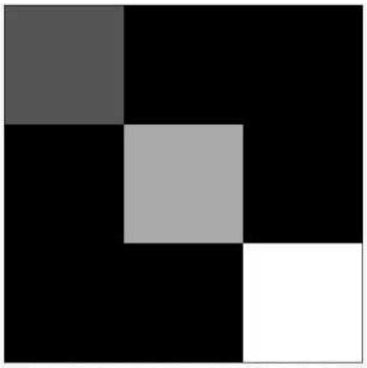
## Description:

Compute an approximation of the `m` largest eigenvalues and eigenvectors of selected symmetric matrices,

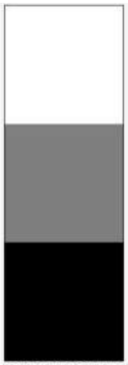
using the Arnoldi iteration method (https://en.wikipedia.org/wiki/Arnoldi_iteration).
A larger `m` goes with better numerical precision.

## Example of use:

```
(1,0,0;0,2,0;0,0,3) +meigen 3
```



[0]: '(1,0,0;0,2,0;0,0,3)' (3x3x1x1)



[1]: '(1,0,0;0,2,0;0,0,3)_c1'
(1x3x1x1)

---

# merge_alpha

**No arguments**

## Description:

Merge selected alpha detail scales into a single image.

Alpha detail scales have been obtained with command **split_alpha**.

# min

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

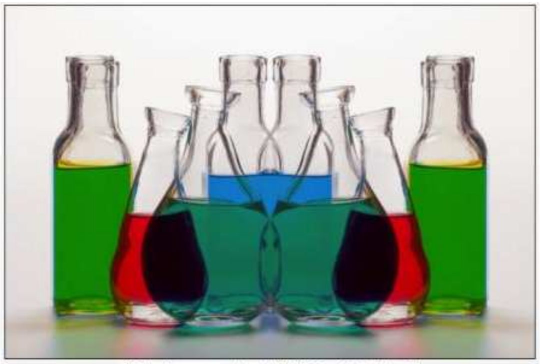
## Description:

Compute the minimum between selected images and specified value, image or mathematical expression, or compute the pointwise minima between selected images.

## Examples of use:

### • Example #1

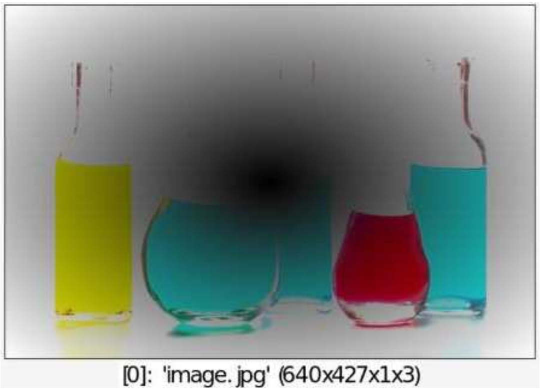
```
image.jpg +mirror x min
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg min 'R=((x/w-0.5)^2+(y/h-0.5)^2)^0.5;255*R'
```



## min_d

**No arguments**

### Description:

Return the minimal depth between selected images.

## min_h

**No arguments**

### Description:

Return the minimal height between selected images.

## min_patch

### Arguments:

- `_patch_size>=1`

### Description:

Return locations of minimal values in local patch-based neighborhood of given size for selected images.

### Default values:

`patch_size=16`.

### Example of use:

```
image.jpg norm +min_patch 16
```

[0]: 'image.jpg' (640x427x1x1)

[1]: 'image_c1.jpg' (640x427x1x1)

## min_s

**No arguments**

### Description:

Return the minimal s size of selected images.

## min_w

**No arguments**

### Description:

Return the minimal width between selected images.

## min_wh

**No arguments**

### Description:

Return the minimal wxh size of selected images.

## min_whd

**No arguments**

### Description:

Return the minimal wxhxd size of selected images.

# min_whds

**No arguments**

## Description:

Return the minimal wxhxdxs size of selected images.

# minabs

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

Compute the minabs between selected images and specified value, image or mathematical expression, or compute the pointwise minabs between selected images.

# minimal_path

## Arguments:

- `x0[%]>=0,y0[%]>=0,z0[%]>=0,x1[%]>=0,y1[%]>=0,z1[%]>=0,_is_high_connectivity={ 0 | 1 }`

## Description:

Compute minimal path between two points on selected potential maps.

## Default values:

`is_high_connectivity=0`.

## Example of use:

```
image.jpg +gradient_norm fill[-1] 1/(1+i) minimal_path[-1]
0,0,0,100%,100%,0 pointcloud[-1] 0 *[-1] 280 to_rgb[-1] ri[-1] [-2],0
or
```

[0]: 'image.jpg' (640x427x1x3)

# mirror

Built-in command

## Arguments:

- `{ x | y | z }...{ x | y | z }`

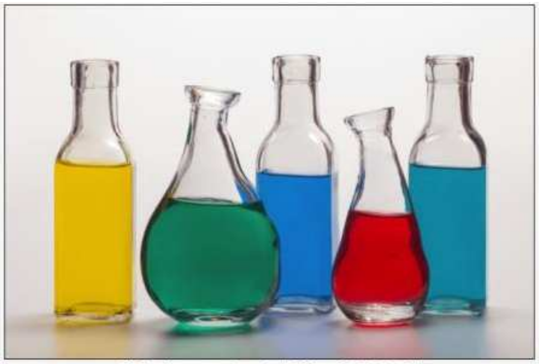
## Description:

Mirror selected images along specified axes.

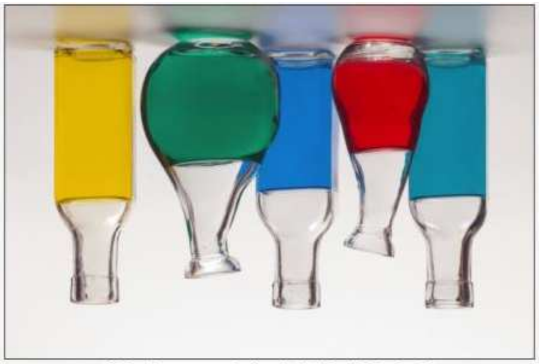
## Examples of use:

### • Example #1

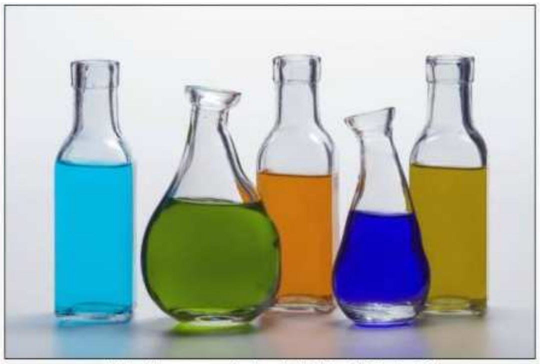
```
image.jpg +mirror y +mirror[0] c
```



[0]: 'image.jpg' (640x427x1x3)



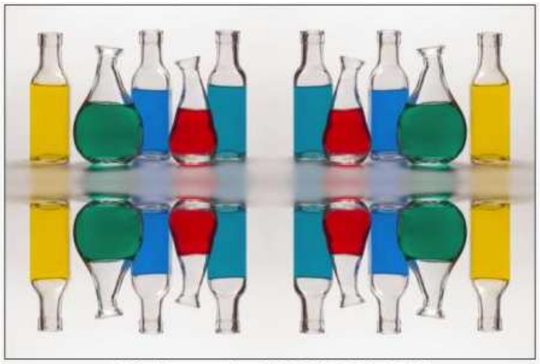
[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg +mirror x +mirror y append_tiles 2,2
```



[0]: 'image.jpg' (1280x854x1x3)

# mix_channels

## Arguments:

- `(a00,...,aMN)` or
- `[matrix]`

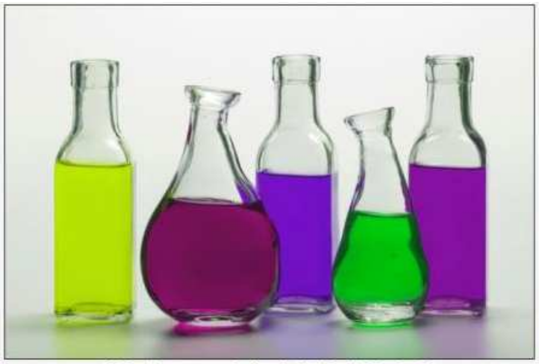
## Description:

Apply specified matrix to channels of selected images.

## Example of use:

```
image.jpg +mix_channels (0,1,0;1,0,0;0,0,1)
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# mix_rgb

## Arguments:

- `a11,a12,a13,a21,a22,a23,a31,a32,a33`

## Description:

Apply 3x3 specified matrix to RGB colors of selected images.

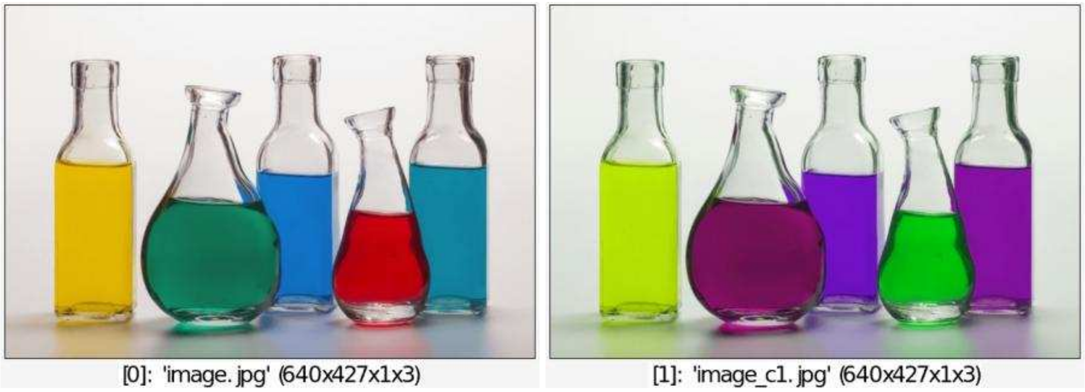
## Default values:

`a11=1`, `a12=a13=a21=0`, `a22=1`, `a23=a31=a32=0` and `a33=1`.

This command has a **tutorial page**.

## Example of use:

```
image.jpg +mix_rgb 0,1,0,1,0,0,0,0,1
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

# mmul

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

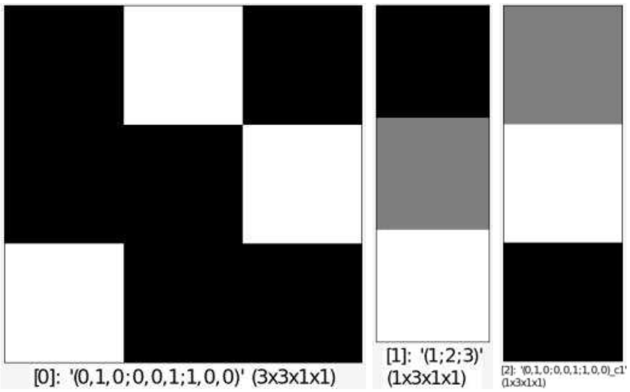
## Description:

Compute the matrix right multiplication of selected matrices/vectors by specified value, image or mathematical expression, or compute the matrix right multiplication of selected images.

(*equivalent to shortcut command* `m*`).

## Example of use:

```
(0,1,0;0,0,1;1,0,0) (1;2;3) +mmul
```


[0]: '(0,1,0;0,0,1;1,0,0)' (3x3x1x1)
[1]: '(1;2;3)' (1x3x1x1)
[2]: '(0,1,0;0,0,1;1,0,0)_c1' (1x3x1x1)

# mod

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

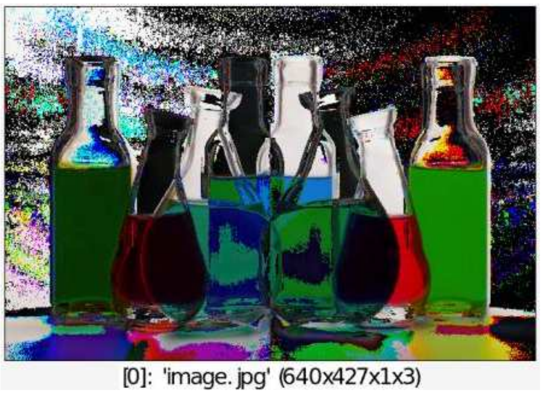
Compute the modulo of selected images with specified value, image or mathematical expression, or compute the pointwise sequential modulo of selected images.

(*equivalent to shortcut command* `%`).

## Examples of use:

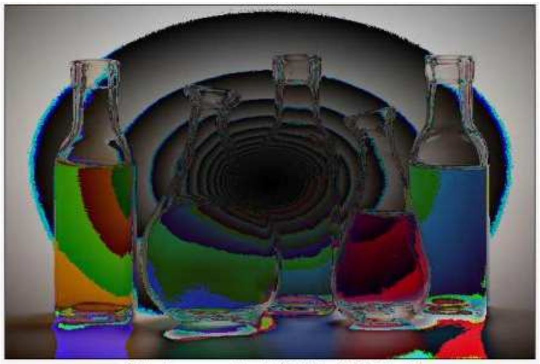
### • Example #1

```
image.jpg +mirror x n. 1,255 round. mod
```



### • Example #2

```
image.jpg mod 'R=((x/w-0.5)^2+(y/h-0.5)^2)^0.5;255*R'
```


[0]: 'image.jpg' (640x427x1x3)

# mode3d

## Arguments:

- `_mode`

## Description:

Set static 3D rendering mode.

(*equivalent to shortcut command* `m3d`).

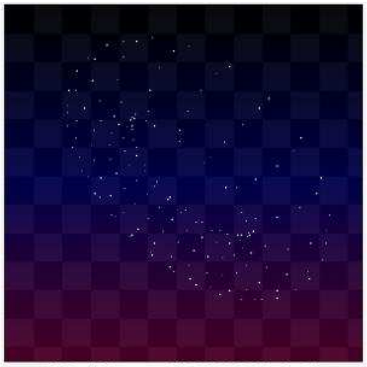
`mode` can be ***{ -1:bounding-box | 0:dots | 1:wireframe | 2:flat | 3:flat-shaded | 4:gouraud-shaded | 5:phong-shaded }***.
Bounding-box mode ( `mode==-1` ) is active only for the interactive 3D viewer.

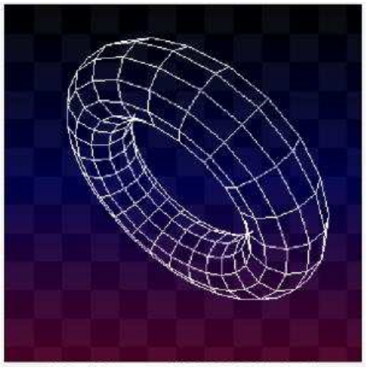
## Default values:

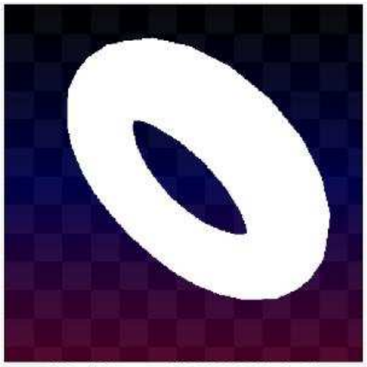
`mode=4`.

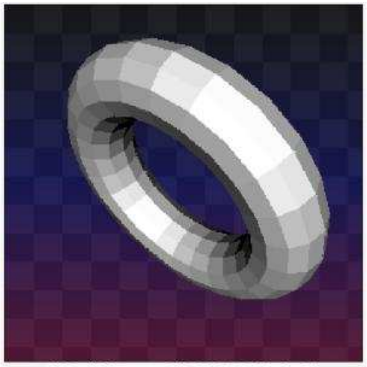
## Example of use:

```
(0,1,2,3,4,5) double3d 0 repeat w { torus3d 100,30 rotate3d[-1]
1,1,0,60 mode3d {0,@$>} snapshot3d[-1] 300 } remove[0]
```

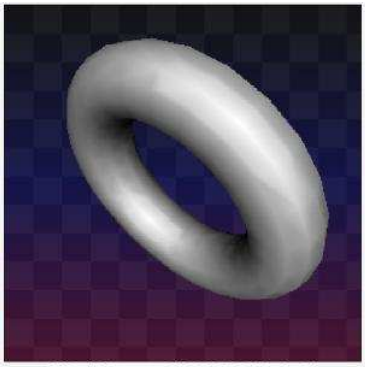

[0]: '[3D torus]' (300x300x1x3)


[1]: '[3D torus]' (300x300x1x3)

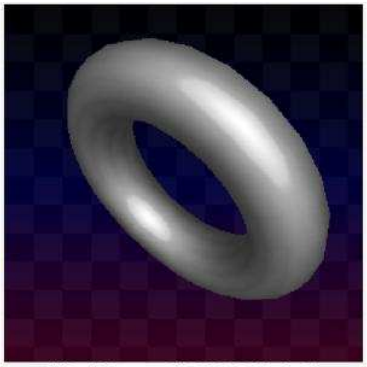

[2]: '[3D torus]' (300x300x1x3)



[3]: '[3D torus]' (300x300x1x3)



[4]: '[3D torus]' (300x300x1x3)



[5]: '[3D torus]' (300x300x1x3)

# moded3d

## Arguments:

- `_mode`

## Description:

Set dynamic 3D rendering mode for interactive 3D viewer.

(*equivalent to shortcut command* `md3d`).

`mode` can be ***{ -1:bounding-box | 0:dots | 1:wireframe | 2:flat | 3:flat-shaded | 4:gouraud-shaded | 5:phong-shaded }***.

## Default values:

`mode=-1`.

# montage

## Arguments:

- `"_layout_code",_montage_mode={ 0<=centering<=1 | 2<=scale+2<=3 },_output_mode={ 0:single layer | 1:multiple layers },"_processing_command"`

## Description:

Create a single image montage from selected images, according to specified layout code :

- `X` to assemble all images using an automatically estimated layout.
- `H` to assemble all images horizontally.
- `V` to assemble all images vertically.
- `A` to assemble all images as an horizontal array.

- `B` to assemble all images as a vertical array.
- `Ha:b` to assemble two blocks `a` and `b` horizontally.
- `Va:b` to assemble two blocks `a` and `b` vertically.
- `Ra` to rotate a block `a` by 90 deg. ( `RRa` for 180 deg. and `RRRa` for 270 deg.).
- `Ma` to mirror a block `a` along the X-axis ( `MRRa` for the Y-axis).

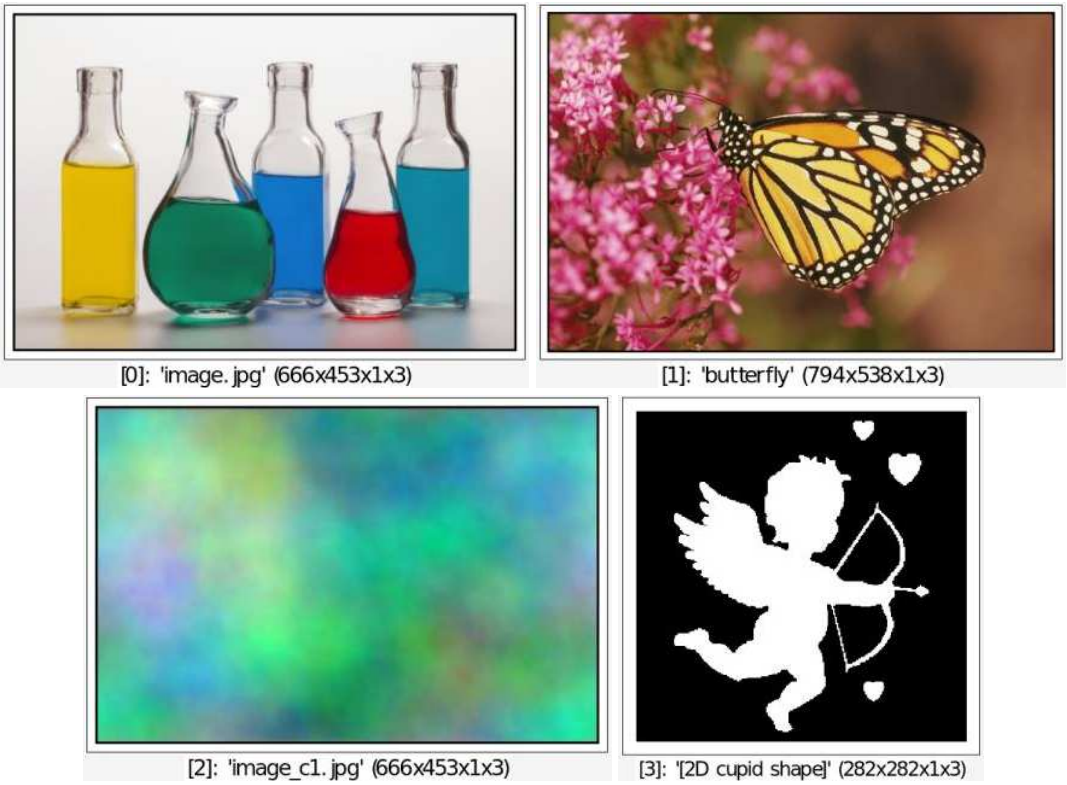
A block `a` can be an image index (treated periodically) or a nested layout expression
`Hb:c`, `Vb:c`, `Rb` or
`Mb` itself.
For example, layout code `H0:V1:2` creates an image where image [0] is on the left, and images [1] and [2]
vertically packed on the right.

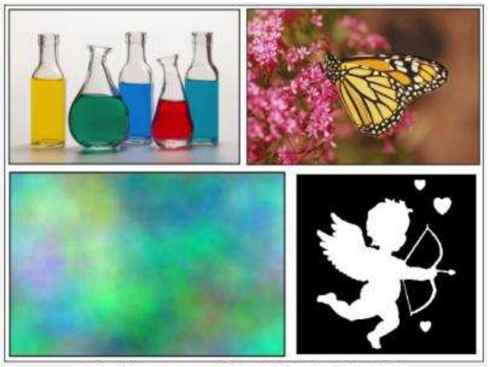
## Default values:

`layout_code=X`, `montage_mode=2`, output_mode='0' and `processing_command=""`.

## Example of use:

```
image.jpg sample ? +plasma[0] shape_cupid 256 normalize 0,255 frame
3,3,0 frame 10,10,255 to_rgb +montage A +montage[^-1] H1:V0:VH2:1H0:3
```


[0]: 'image.jpg' (666x453x1x3)
[1]: 'butterfly' (794x538x1x3)
[2]: 'image_c1.jpg' (666x453x1x3)
[3]: '[2D cupid shape]' (282x282x1x3)



[4]: '[Montage 'A']_c1' (697x518x1x3)

[5]: '[Montage 'H1:V0:VH2:1H0:3']_c1' (1172x538x1x3)

# morph

## Arguments:

- `nb_inner_frames>=1,_smoothness>=0,_precision>=0`

## Description:

Create morphing sequence between selected images.

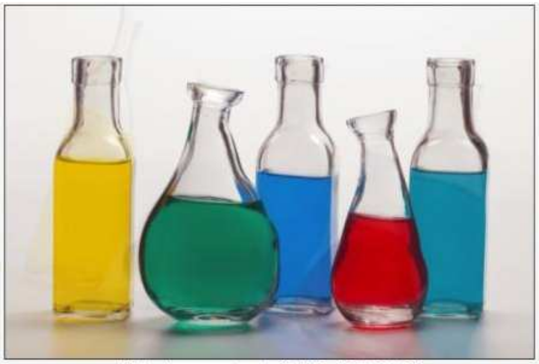
## Default values:

`smoothness=0.1` and `precision=4`.

## Example of use:

```
image.jpg +rotate 20,1,1,50%,50% morph 9
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image.jpg' (640x427x1x3)

[2]: 'image.jpg' (640x427x1x3)

[3]: 'image.jpg' (640x427x1x3)

[4]: 'image.jpg' (640x427x1x3)

[5]: 'image.jpg' (640x427x1x3)

[6]: 'image.jpg' (640x427x1x3)

[7]: 'image.jpg' (640x427x1x3)

[8]: 'image.jpg' (640x427x1x3)

[9]: 'image.jpg' (640x427x1x3)

# morph_files

## Arguments:

- `"filename_pattern",_nb_inner_frames>0,_smoothness>=0,_precision>=0,_first_fra >=0 | -1=last },_frame_step>=1,_output_filename`

## Description:

Generate a temporal morphing from specified input image files, in a streamed way.

If a display window is opened, rendered frames are displayed in it during processing.
The output filename may have extension `.avi` or `.mp4` (saved as a video), or any other usual image
file extension (saved as a sequence of images).

## Default values:

`nb_inner_frames=10`, `smoothness=0.1`, `precision=4`, `first_frame=0`, `last_frame=-1`, `frame_step=1` and `output_filename=(undefined)`.

# morph_rbf

## Arguments:

- `nb_inner_frames>=1,xs0[%],ys0[%],xt0[%],yt0[%],...,xsN[%],ysN[%],xtN[%],ytN[%`

## Description:

Create morphing sequence between selected images, using RBF-based interpolation.

Each argument (xsk,ysk)-(xtk,ytk) corresponds to the coordinates of a keypoint
respectively on the source and target images. The set of all keypoints define the overall image deformation.

# morph_video

## Arguments:

- `video_filename,_nb_inner_frames>0,_smoothness>=0,_precision>=0,_first_frame>=`
`>=0 | -1=last },_frame_step>=1,_output_filename`

## Description:

Generate a temporal morphing from specified input video file, in a streamed way.

If a display window is opened, rendered frames are displayed in it during processing.
The output filename may have extension `.avi` or `.mp4` (saved as a video), or any other usual image
file extension (saved as a sequence of images).
This command requires features from the OpenCV library (not enabled in G'MIC by default).

## Default values:

`nb_inner_frames=10`, `smoothness=0.1`, `precision=4`, `first_frame=0`, `last_frame=-1`, `frame_step=1` and `output_filename=(undefined)`.

---

# mosaic

## Arguments:

- `0<=_density<=100`

## Description:

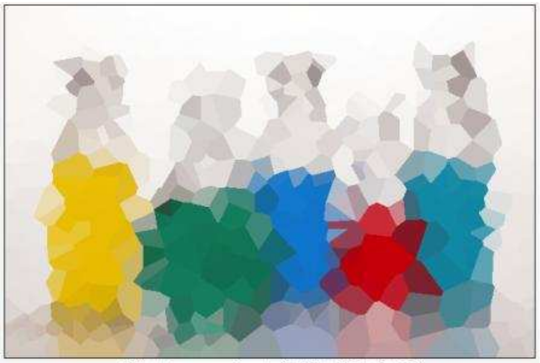
Create random mosaic from selected images.

## Default values:

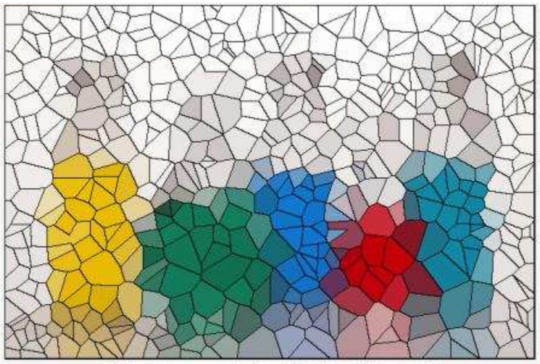
`density=30`.

## Example of use:

```
image.jpg mosaic , +fill "I!=J(1) || I!=J(0,1)?[0,0,0]:I"
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# move

Built-in command

## Arguments:

- `position[%]`

## Description:

Move selected images at specified position.

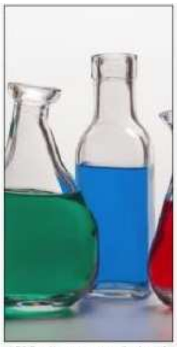
Images are actually inserted between current positions `position-1` and `position`.

(*equivalent to shortcut command* `mv`).

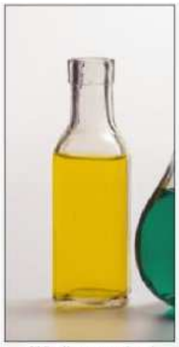
## Examples of use:

### • Example #1

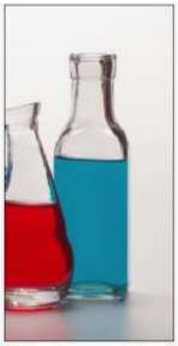
```
image.jpg split x,3 move[1] 0
```



[0]: 'image_c1.jpg' (213x427x1x3)



[1]: 'image.jpg' (214x427x1x3)



[2]: 'image_c2.jpg' (213x427x1x3)

### • Example #2

```
image.jpg split x move[50%--1:2] 0 append x
```

[0]: 'image_c320.jpg' (640x427x1x3)

## mproj

Built-in command

### Arguments:

- `[dictionary],_method,_max_iter={ 0:auto | >0 },_max_residual>=0`

### Description:

Find best matching projection of selected matrices onto the span of an over-complete

dictionary D, using the orthogonal projection or Matching Pursuit algorithm.
Selected images are 2D-matrices in which each column represent a signal to project.
`[dictionary]` is a matrix in which each column is an element of the dictionary D.
`method` tells what projection algorithm must be applied. It can be:
 - 0 = orthogonal projection (least-squares solution using LU-based solver).
 - 1 = matching pursuit.
 - 2 = matching pursuit, with a single orthogonal projection step at the end.
 - >=3 = orthogonal matching pursuit where an orthogonal projection step is performed
 every `method-2` iterations.
`max_iter` sets the max number of iterations processed for each signal.
If set to `0` (default), `max_iter` is equal to the number of columns in D.
(only meaningful for matching pursuit and its variants).
`max_residual` gives a stopping criterion on signal reconstruction accuracy.
(only meaningful for matching pursuit and its variants).
For each selected image, the result is returned as a matrix W
whose columns correspond to the weights associated to each column of D,
such that the matrix product D*W is an approximation of the input matrix.

### Default values:

`method=0`, `max_iter=0` and `max_residual=1e-6`.

## mse

### Arguments:

- `[reference]`

## Description:

Return the MSE (Mean-Squared Error) between selected images and specified reference image.

This command does not modify the images. It returns a value or a list of values in the status.

# mse_matrix

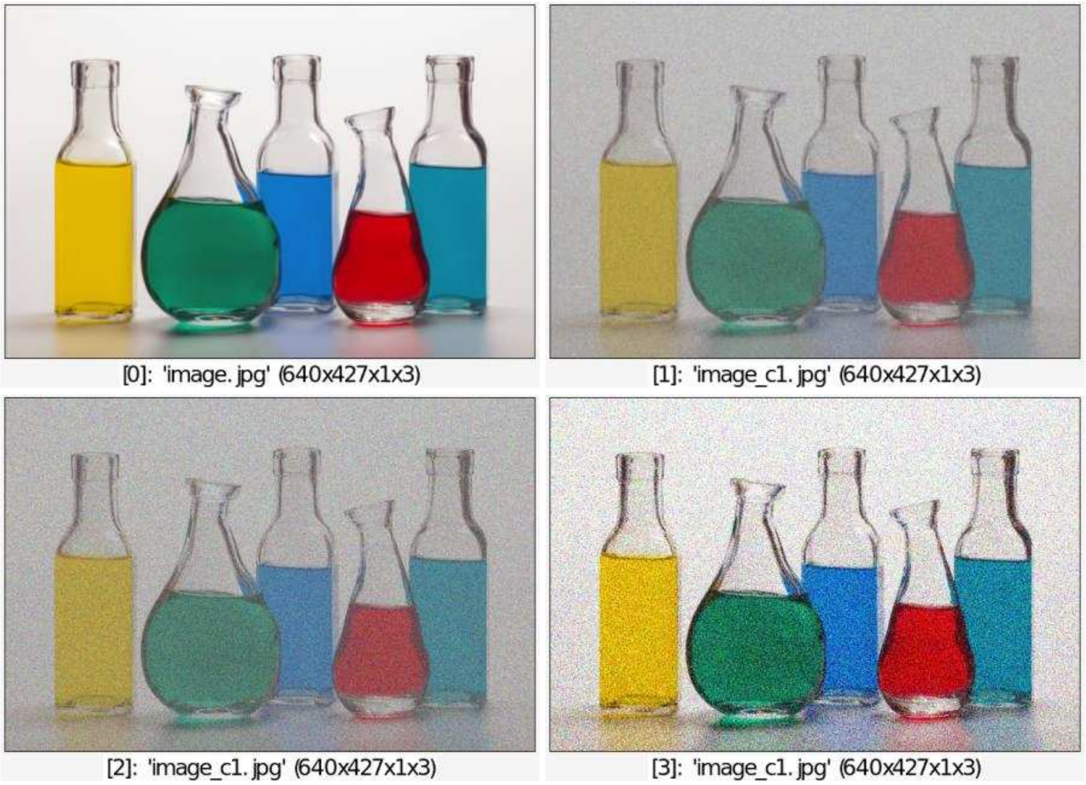
### No arguments

## Description:

Compute MSE (Mean-Squared Error) matrix between selected images.

## Example of use:

```
image.jpg +noise 30 +noise[0] 35 +noise[0] 38 cut. 0,255 +mse_matrix
```

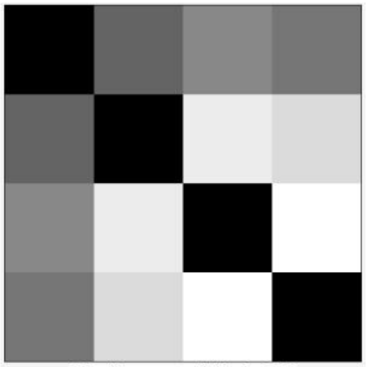


[0]: 'image. jpg' (640x427x1x3)

[1]: 'image_c1. jpg' (640x427x1x3)

[2]: 'image_c1. jpg' (640x427x1x3)

[3]: 'image_c1. jpg' (640x427x1x3)



[4]: '[unnamed]' (4x4x1x1)

# mul

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

Multiply selected images by specified value, image or mathematical expression, or compute the pointwise product of selected images.

(*equivalent to shortcut command* `*`).

## See also:

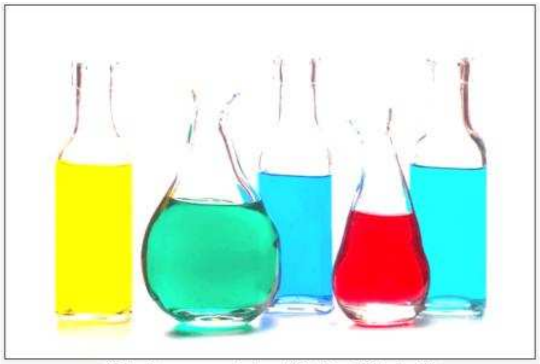
**add**, **sub**, **div**.

## Examples of use:

### • Example #1

```
image.jpg +mul 2 cut 0,255
```

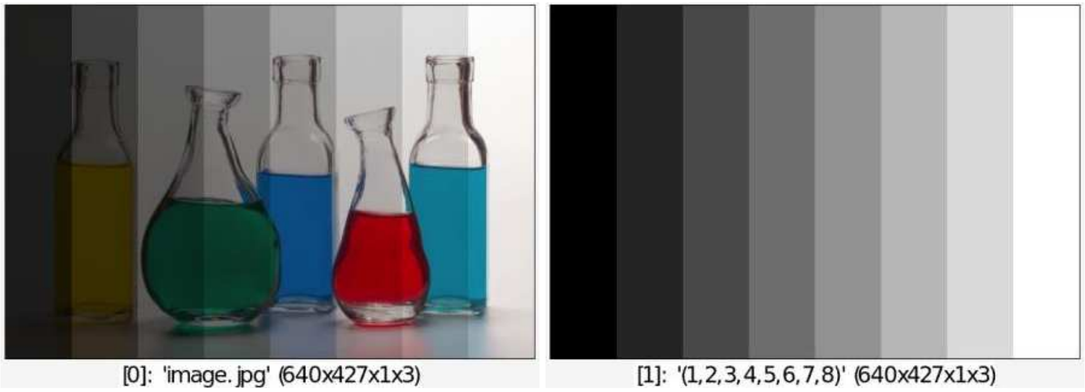
[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

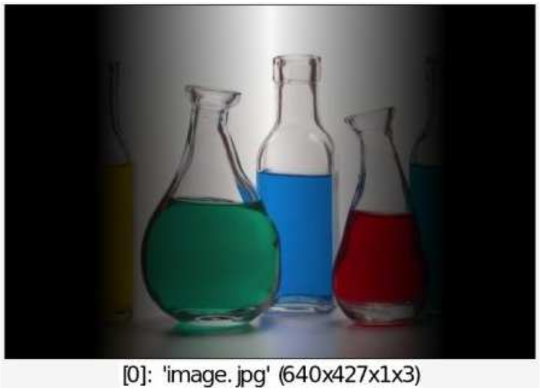
```
image.jpg (1,2,3,4,5,6,7,8) ri[-1] [0] mul[0] [-1]
```



[0]: 'image.jpg' (640x427x1x3)    [1]: '(1,2,3,4,5,6,7,8)' (640x427x1x3)

- **Example #3**

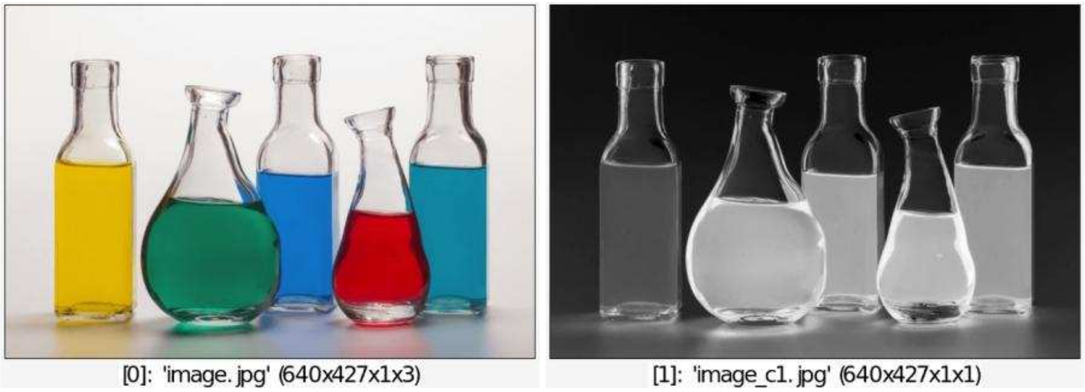
```
image.jpg mul '1-3*abs(x/w-0.5)' cut 0,255
```



[0]: 'image.jpg' (640x427x1x3)

- **Example #4**

```
image.jpg +luminance negate[-1] +mul
```



[0]: 'image.jpg' (640x427x1x3)    [1]: 'image_c1.jpg' (640x427x1x1)

[2]: 'image_c1.jpg' (640x427x1x3)

## mul3d

Built-in command

### Arguments:

- `factor` or
- `factor_x,factor_y,_factor_z`

### Description:

Scale selected 3D objects isotropically or anisotropically, with specified factors.

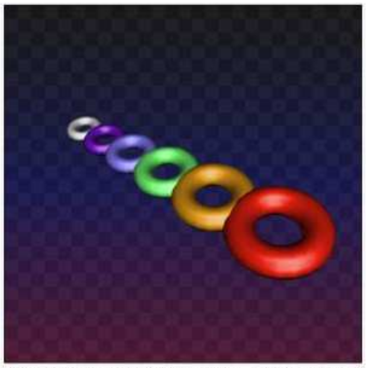
(*equivalent to shortcut command* `*3d`).

### Default values:

`factor_z=1`.

### Example of use:

```
torus3d 5,2 repeat 5 { +add3d[-1] 10,0,0 mul3d[-1] 1.2 color3d[-1] $
{-rgb} } add3d
```



[0]: '[3D torus]' (1728 vert., 1728 prim.)

## mul_channels

### Arguments:

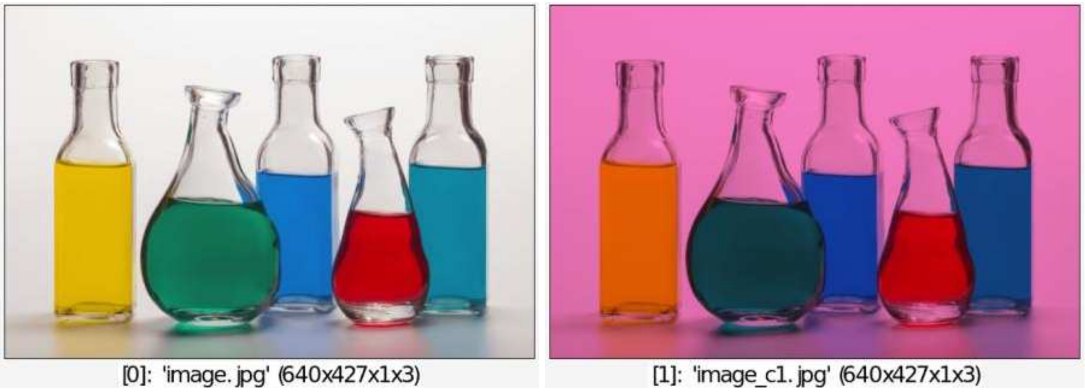
- `value1,_value2,...,_valueN`

### Description:

Multiply channels of selected images by specified sequence of values.

### Example of use:

```
image.jpg +mul_channels 1,0.5,0.8
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

## mul_complex

### Arguments:

- `[multiplier_real,multiplier_imag]`

### Description:

Perform multiplication of the selected complex pairs (real1,imag1,...,realN,imagN) of images by

specified complex pair of images (multiplier_real,multiplier_imag).
In complex pairs, the real image must be always located before the imaginary image in the image list.

## mutex

**Built-in command**

### Arguments:

- `index,_action={ 0:unlock | 1:lock }`

### Description:

Lock or unlock specified mutex for multi-threaded programming.

A locked mutex can be unlocked only by the same thread. All mutexes are unlocked by default. `index` designates the mutex index, in [0,255].

## Default values:

`action=1`.

---

# nadirzenith2equirectangular

**No arguments**

## Description:

Transform selected nadir/zenith rectilinear projections to equirectangular images.

---

# name

**Built-in command**

## Arguments:

- `"name1","name2",...,"nameN"`

## Description:

Set names of selected images.

- If no explicit image selection is given, image selection is assumed to be `[-N--1]`, where `N` is the number of specified arguments.
- If `N` is higher than the number of images in selection, an error is thrown.
- If `N` is lower than the number of images in selection, image names are assigned in a periodic way, i.e. `name(selection[k]) = arg[k%N]`.

(*equivalent to shortcut command* `=>`).

This command has a **tutorial page**.

## Example of use:

```
image.jpg name image blur[image] 2
```

[0]: 'image' (640x427x1x3)

## name2color

### Arguments:

- `name`

### Description:

Return the R,G,B color that matches the specified color name.

## named

Built-in command

### Arguments:

- `_mode,"name1","name2",...`

### Description:

Return the set of indices corresponding to images of the selection with specified names.

After this command returns, the status contains a list of indices (unsigned integers), separated by commas (or an empty string if no images with those names have been found).

(*equivalent to shortcut command* `nmd`).

`mode` can be ***{ 0:all indices (default) | 1:lowest index | 2:highest index | 3:all indices (case insensitive) | 4:lowest index (case insensitive) | 5:highest index (case insensitive)}***

## narg

### Arguments:

- `arg1,arg2,...,argN`

### Description:

Return number of specified arguments.

---

## negate

### Arguments:

- `base_value` or
- `(no arg)`

### Description:

Negate image values.

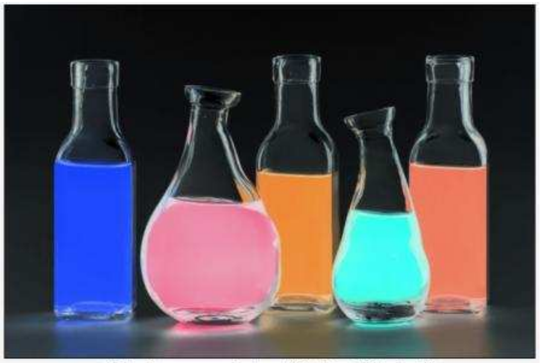
### Default values:

`base_value=(undefined)`.

### Example of use:

```
image.jpg +negate
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

## neq

Built-in command

### Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

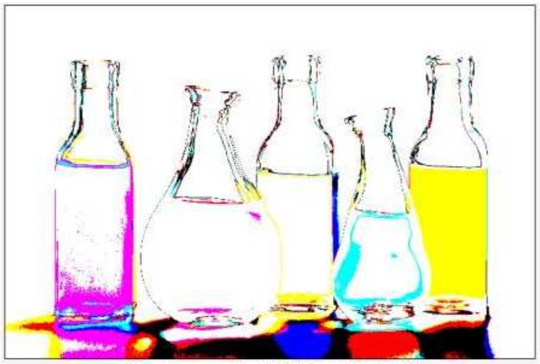
### Description:

Compute the boolean inequality of selected images with specified value, image or mathematical expression, or compute the boolean inequality of selected images.

(*equivalent to shortcut command* `!=`).

## Example of use:

```
image.jpg round 40 neq {round(ia,40)}
```



[0]: 'image.jpg' (640x427x1x3)

# network

**Built-in command**

## Arguments:

- `mode={ -1=disabled | 0:enabled w/o timeout | >0:enabled w/ specified timeout in seconds }`

## Description:

Enable/disable load-from-network and set corresponding timeout.

(Default mode is 'enabled w/o timeout').

# newton_fractal

## Arguments:

- `z0r,z0i,z1r,z1i,_angle,0<=_descent_method<=2,_iteration_max>=0,_convergence_precision>0,_expr_p(z),_`

## Description:

Draw newton fractal on selected images, for complex numbers in range (z0r,z0i) - (z1r,z1i).

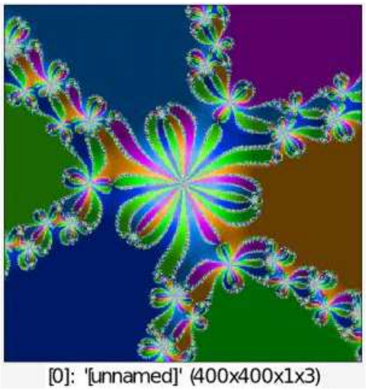
Resulting images have 3 channels whose meaning is [ last_zr, last_zi, nb_iter_used_for_convergence ].
`descent_method` can be ***`{ 0:secant | 1:newton | 2:householder }`***.

## Default values:

angle=0, descent_method=1, iteration_max=200, convergence_precision=0.01, expr_p(z)=z^^3-1, expr_dp(z)=3*z^^2 and expr_d2z(z)=6*z.

### Example of use:

```
400,400 newton_fractal -1.5,-1.5,1.5,1.5,0,2,200,0.01,"z^^6 + z^^3 - 1","6*z^^5 + 3*z^^2","30*z^^4 + 6*z" f "[ atan2(i1,i0)*90+20,1,cut(i2/30,0.2,0.7) ]" hsl2rgb
```



[0]: '[unnamed]' (400x400x1x3)

---

# nlmeans

### Arguments:

- `[guide],_patch_radius>0,_spatial_bandwidth>0,_tonal_bandwidth>0,_patch_measur`
  or
- `_patch_radius>0,_spatial_bandwidth>0,_tonal_bandwidth>0,_patch_measure_commar`

### Description:

Apply non local means denoising of Buades et al, 2005. on selected images.

The patch is a gaussian function of std_patch_radius.
The spatial kernel is a rectangle of radius spatial_bandwidth.
The tonal kernel is exponential (exp(-d^2/_tonal_bandwidth^2))
with d the euclidean distance between image patches.

### Default values:

patch_radius=4, spatial_bandwidth=4, tonal_bandwidth=10 and patch_measure_command=-norm.

### Example of use:

```
image.jpg +noise 10 nlmeans[-1] 4,4,{0.6*${-std_noise}}
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

---

# nlmeans_core

## Arguments:

- `_reference_image,_scaling_map,_patch_radius>0,_spatial_bandwidth>0`

## Description:

Apply non local means denoising using a image for weight and a map for scaling

---

# nn_add

## Arguments:

- `out,in0,_in1`

## Description:

Add an `add` layer to the network.

## Default values:

'in1=. (previous layer)'.

---

# nn_append

## Arguments:

- `out,in0,_in1`

## Description:

Add an `append` layer to the network.

### Default values:

'in1=. (previous layer)'.

---

# nn_avgpool2d

### Arguments:

- `out,_in,_patch_size>1`

### Description:

Add a `avgpool2d` layer (2D average pooling) to the network.

### Default values:

'in=. (previous layer)'.

---

# nn_avgpool3d

### Arguments:

- `out,_in,_patch_size>1`

### Description:

Add a `avgpool3d` layer (3D average pooling) to the network.

### Default values:

'in=. (previous layer)'.

---

# nn_check_layer

### Arguments:

- `name`

### Description:

Check that the layer with specified name already exists in the network.

---

# nn_clone

## Arguments:

- `name0,name1,_in`

## Description:

Add a `clone` layer to the network.

## Default values:

'in=. (previous layer)'.

---

# nn_conv2d

## Arguments:

- `out,in,nb_channels>0,_kernel_size>0,_stride>0,_dilation,_border_shrink>=0,_bo`

## Description:

Add a `conv2d` layer (2D convolutional layer) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

## Default values:

`kernel_size=3`, `stride=1`, `dilation=1`, `border_shrink=0`, `boundary_conditions=1` and `learning_mode=3`.

---

# nn_conv2dnl

## Arguments:

- `out,in,nb_channels>0,_kernel_size>0,_stride>0,_dilation>0,_border_shrink>=0,_`

## Description:

Add a `conv2dnl` (2D convolutional layer followed by a non-linearity) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only |***

***3:weights+biases }***.

### Default values:

`kernel_size=3`, `stride=1`, `dilation=1`, `border_shrink=0`, `boundary_conditions=1`, `activation=leakyrelu` and `learning_mode=3`.

---

## nn_conv2dnnl

### Arguments:

- `out,in,nb_channels>0,_kernel_size>0,_stride>0,_dilation>0,_border_shrink>=0,_`

### Description:

Add a `conv2dnnl` (2D convolutional layer followed by a normalization layer, then a non-linearity) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

`kernel_size=3`, `stride=1`, `dilation=1`, `border_shrink=0`, `boundary_conditions=1`, `activation=leakyrelu` and `learning_mode=3`.

---

## nn_conv3d

### Arguments:

- `out,in,nb_channels>0,_kernel_size>0,_stride>0,_dilation,_border_shrink>=0,_bo`

### Description:

Add a `conv3d` layer (3D convolutional layer) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

`kernel_size=3`, `stride=1`, `dilation=1`, `border_shrink=0`, `boundary_conditions=1` and `learning_mode=3`.

## nn_conv3dnl

### Arguments:

- `out,in,nb_channels>0,_kernel_size>0,_stride>0,_dilation>0,_border_shrink>=0,_`

### Description:

Add a `conv3dnl` (3D convolutional layer followed by a non-linearity) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

`kernel_size=3`, `stride=1`, `dilation=1`, `border_shrink=0`, `boundary_conditions=1`, `activation=leakyrelu` and `learning_mode=3`.

## nn_conv3dnnl

### Arguments:

- `out,in,nb_channels>0,_kernel_size>0,_stride>0,_dilation>0,_border_shrink>=0,_`

### Description:

Add a `conv3dnnl` (3D convolutional layer followed by a normalization layer, then a non-linearity) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

`kernel_size=3`, `stride=1`, `dilation=1`, `border_shrink=0`, `boundary_conditions=1`, `activation=leakyrelu` and `learning_mode=3`.

## nn_crop

### Arguments:

- `out,in,x0,y0,z0,c0,x1,y1,z1,c1,_boundary_conditions`

### Description:

Add a `crop` layer to the network.

`boundary_conditions` can be ***`{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`***.

### Default values:

`boundary_conditions=0`.

---

## nn_distance

### Arguments:

- `out,in0,_in1,_metric={ 0:squared-L2 | p>0:Lp-norm }`

### Description:

Add a `distance` layer to the network (distance between two inputs, with specified metric).

### Default values:

'in=. (previous layer)',

---

## nn_dropout

### Arguments:

- `out,in,0<=dropout_rate<1`

### Description:

Add a `dropout` layer to the network.

---

## nn_fc

### Arguments:

- `out,in,nb_channels>0,0<=_learning_mode<=3`

### Description:

Add a `fc` layer (fully connected layer) to the network.

`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

`learning_mode=3`.

---

# nn_fcnl

### Arguments:

- `out,in,nb_neurons>0,_activation,0<=_learning_mode<=3`

### Description:

Add a `fcnl` layer (fully connected layer followed by a non-linearity) to the network.

`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

`activation=leakyrelu` and `learning_mode=3`.

---

# nn_fcnnl

### Arguments:

- `out,in,nb_neurons>0,_activation,0<=_learning_mode<=3`

### Description:

Add a `fcnnl` layer (fully connected layer followed by a normalization layer, then a non-linearity) to the network.

`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

`activation=leakyrelu` and `learning_mode=3`.

---

# nn_init

### No arguments

## Description:

Initialize a new network.

---

# nn_input

## Arguments:

- `name,width,_height,_depth,_spectrum`

## Description:

Add a new `input` to the network.

## Default values:

`height=1`, `depth=1` and `spectrum=1`.

---

# nn_lib

### No arguments

## Description:

Return the list of library functions that has to be included in a math expression,in order to use the neural network library.

---

# nn_load

## Arguments:

- `'filename.gmz',_include_trainer_data={ 0:no | 1:yes }`

## Description:

Load and initialize network saved as a .gmz file.

Neural network files can be only loaded in .gmz format.

## Default values:

`include_trainer_data=1`.

---

# nn_loss_binary_crossentropy

## Arguments:

- `out,in,ground_truth`

## Description:

Add a `binary_crossentropy` loss to the network (binary cross entropy).

---

# nn_loss_crossentropy

## Arguments:

- `out,in,ground_truth`

## Description:

Add a `crossentropy` loss to the network (cross entropy).

---

# nn_loss_mse

## Arguments:

- `out,in,ground_truth`

## Description:

Add a `mse` loss to the network (mean-squared error).

---

# nn_loss_normp

## Arguments:

- `out,in,ground_truth,_metric={ 0:squared-L2 | p>0:Lp-norm }`

## Description:

Add a `normp` loss to the network (||out - ground_truth||_p).

## Default values:

`metric=1`.

---

# nn_loss_softmax_crossentropy

## Arguments:

- `out,in,ground_truth`

## Description:

Add a `softmax_crossentropy` loss to the network (softmax followed by cross entropy).

---

# nn_maxpool2d

## Arguments:

- `out,_in,_patch_size>1`

## Description:

Add a `maxpool2d` layer (2D max pooling) to the network.

## Default values:

'in=. (previous layer)' and `patch_size=2`.

---

# nn_maxpool3d

## Arguments:

- `out,_in,_patch_size>1`

## Description:

Add a `maxpool3d` layer (3d max pooling) to the network.

## Default values:

'in=. (previous layer)' and `patch_size=2`.

---

# nn_mul

### Arguments:

- `out,in0,_in1`

### Description:

Add an `mul` layer to the network.

### Default values:

'in1=. (previous layer)'.

---

## nn_nl

### Arguments:

- `out,_in,_activation`

### Description:

Add a `nl` (nonlinearity) layer to the network.

`activation` can be ***{ elu | gelu | leakyrelu | linear | relu | sigmoid | sin | sinc | softmax | sqr | sqrt | swish | tanh }***.

### Default values:

'in=. (previous layer)' and `activation=leakyrelu`.

---

## nn_nlfc

### Arguments:

- `out,in,nb_channels>0,_activation,0<=_learning_mode<=3`

### Description:

Add a `nlfc` layer (nonlinear fully connected layer) to the network.

`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

`activation=leakyrelu` and `learning_mode=3`.

---

# nn_normalize

## Arguments:

- `out,_in,_normalization_mode_,0<=_learning_mode<=3`

## Description:

Add a `normalize` layer to the network.

`normalization_mode` can be ***{ 0:global parameters | 1:channel-by-channel parameters }***
`learning_mode` can be ***{ 0:no learning | 1:alpha only | 2:beta only | 3:alpha+beta }***

## Default values:

'in=. (previous layer)`,'normalization_mode=0` and `learning_mode=3`.

---

# nn_patchdown2d

## Arguments:

- `out,_in,_patch_size>1`

## Description:

Add a `patchdown2d` (2D downscale by patch) layer to the network.

## Default values:

'in=. (previous layer)' and `patch_size=2`.

---

# nn_patchdown3d

## Arguments:

- `out,_in,_patch_size>1`

## Description:

Add a `patchdown3d` (3D downscale by patch) layer to the network.

## Default values:

'in=. (previous layer)' and `patch_size=2`.

# nn_patchup2d

## Arguments:

- `out,_in,_patch_size>1`

## Description:

Add a `patchup2d` (2D upscale by patch) layer to the network.

## Default values:

'in=. (previous layer)' and `patch_size=2`.

# nn_patchup3d

## Arguments:

- `out,_in,_patch_size>1`

## Description:

Add a `patchup3d` (3D upscale by patch) layer to the network.

## Default values:

'in=. (previous layer)' and `patch_size=2`.

# nn_print

### No arguments

## Description:

Print info on current neural network.

# nn_rename

## Arguments:

- `out,_in`

## Description:

Add a `rename` layer to the network.

## Default values:

'in=. (previous layer)'.

---

# nn_resconv2d

## Arguments:

- `out,_in,_kernel_size>0,_dilation>0,_boundary_conditions,0<=_learning_mode<=3`

## Description:

Add a `resconv2d` (residual 2D convolutional layer) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

## Default values:

'in=. (previous layer)', `kernel_size=3`, `dilation=1`, `boundary_conditions=1` and `learning_mode=3`.

---

# nn_resconv2dnl

## Arguments:

- `out,_in,_kernel_size>0,_dilation>0,_boundary_conditions,_activation,0<=_learn`

## Description:

Add a `resconv2dnl` (residual 2D convolutional layer followed by a non-linearity) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

## Default values:

'in=. (previous layer)', `kernel_size=3`, `dilation=1`, `boundary_conditions=1`, `activation=leakyrelu` and `learning_mode=3`.

## nn_resconv2dnnl

### Arguments:

- `out,_in,_kernel_size>0,_dilation>0,_boundary_conditions,_activation,0<=_learn`

### Description:

Add a `resconv2dnnl` (residual 2D convolutional layer followed by a normalization layer, then a non-linearity)to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

'in=. (previous layer)', `kernel_size=3`, `dilation=1`, `boundary_conditions=1`, `activation=leakyrelu` and `learning_mode=3`.

## nn_resconv3d

### Arguments:

- `out,_in,_kernel_size>0,_dilation>0,_boundary_conditions,0<=_learning_mode<=3`

### Description:

Add a `resconv3d` (residual 3D convolutional layer) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

### Default values:

'in=. (previous layer)', `kernel_size=3`, `dilation=1`, `boundary_conditions=1` and `learning_mode=3`.

## nn_resconv3dnl

### Arguments:

- `out,_in,_kernel_size>0,_dilation>0,_boundary_conditions,_activation,0<=_learn`

## Description:

Add a `resconv3dnl` (residual 3D convolutional layer followed by a non-linearity) to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

## Default values:

'in=. (previous layer)', `kernel_size=3`, `dilation=1`, `boundary_conditions=1`, activation='leakyrelu' and `learning_mode=3`.

---

# nn_resconv3dnnl

## Arguments:

- `out,_in,_kernel_size>0,_dilation>0,_boundary_conditions,_activation,0<=_learn`

## Description:

Add a `resconv3dnnl` (residual 3D convolutional layer followed by a normalization layer, then a non-linearity)to the network.

`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
`learning_mode` can be ***{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }***.

## Default values:

'in=. (previous layer)', `kernel_size=3`, `dilation=1`, `boundary_conditions=1`, `activation=leakyrelu` and `learning_mode=3`.

---

# nn_resfc

## Arguments:

- `out,_in,0<=_learning_mode<=3`

## Description:

Add a `resfc` (residual fully connecter layer) to the network.

`learning_mode` can be ***`{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }`***.

### Default values:

'in=. (previous layer)' and `learning_mode=3`.

---

# nn_resfcnl

### Arguments:

- `out,_in,_activation,0<=_learning_mode<=3`

### Description:

Add a `resfcnl` (residual fully connecter layer followed by a non-linearity) to the network.

`learning_mode` can be ***`{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }`***.

### Default values:

'in=. (previous layer)', `activation=leakyrelu` and `learning_mode=3`.

---

# nn_resfcnnl

### Arguments:

- `out,_in,_activation,0<=_learning_mode<=3`

### Description:

Add a `resfcnnl` (residual fully connecter layer followed by a normalization layer, then a non-linearity) to the network.

`learning_mode` can be ***`{ 0:no learning | 1:weights only | 2:biases only | 3:weights+biases }`***.

### Default values:

'in=. (previous layer)', `activation=leakyrelu` and `learning_mode=3`.

---

# nn_reshape

## Arguments:

- `out,in,width>0,height>0,depth>0,spectrum>0`

## Description:

Add a `reshape` layer to the network.

---

# nn_resize

## Arguments:

- `out,in,width[%]>0,_height[%]>0,_depth[%]>0,_spectrum[%]>0,_interpolation`

## Description:

Add a `resize` layer to the network.

## Default values:

`height=depth=spectrum=100%` and `interpolation=3`.

---

# nn_run

## Arguments:

- `out,in,"command",_width[%]>0,_height[%]>0,_depth[%]>0,_spectrum[%]>0`

## Description:

Add a `run` layer to the network.

## Default values:

`width=height=depth=spectrum=100%`.

---

# nn_save

## Arguments:

- `'filename.gmz',_include_trainer_data={ 0:no | 1:yes }`

## Description:

Save current network as a .gmz file.

`.gmz` is mandatory extension, specifying another file extension will throw an error.

### Default values:

`include_trainer_data=1`.

---

## nn_size

**No arguments**

### Description:

Return size of the current network (i.e. number of stored parameters).

---

## nn_split

### Arguments:

- `name0,name1,in,nb_channels0`

### Description:

Add a `split` layer to the network.

---

## nn_store

### Arguments:

- `'variable_name',_include_trainer_data={ 0:no | 1:yes }`

### Description:

Store current network into a variable.

### Default values:

`include_trainer_data=1`.

---

## nn_trainer

### Arguments:

- `name,_loss,_learning_rate>0,_optimizer,_scheduler`

## Description:

Add a network trainer to the network.

`optimizer` can be *{ sgd | rmsprop | adam | adamax }*.
`scheduler` can be *{ constant | linear | exponential | adaptive }*.

## Default values:

'loss=. (previous loss)', `learning_rate=2e-4`, `optimizer=rmsprop` and `scheduler=constant`.

---

# noarg

Built-in command

**No arguments**

## Description:

Used in a custom command, `noarg` tells the command that its argument list have not been used

finally, and so they must be evaluated next in the G'MIC pipeline, just as if the custom command takes no arguments at all.
Use this command to write a custom command which can decide if it takes arguments or not.

---

# noise

Built-in command

## Arguments:

- `amplitude>=0[%],_noise_type`

## Description:

Add random noise to selected images.

`noise_type` can be *{ 0:gaussian | 1:uniform | 2:salt&pepper | 3:poisson | 4:rice }*.

## Default values:

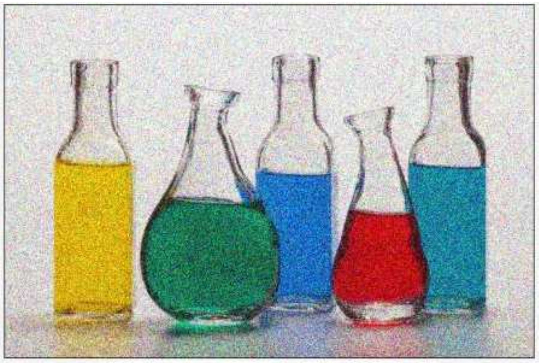
`noise_type=0`.

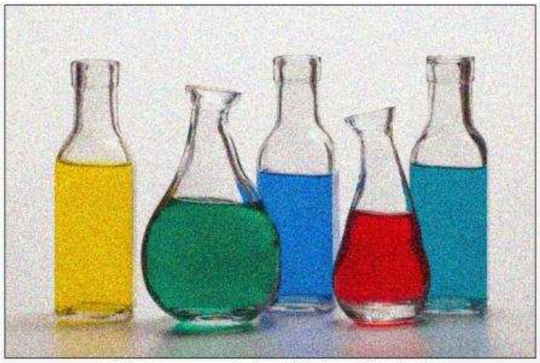
## Examples of use:

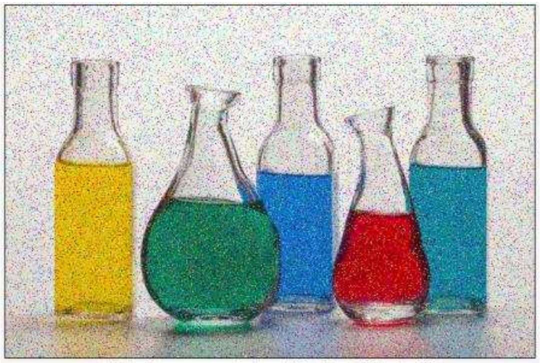
**• Example #1**

```
image.jpg +noise[0] 50,0 +noise[0] 50,1 +noise[0] 10,2 cut 0,255
```

[0]: 'image. jpg' (640x427x1x3)

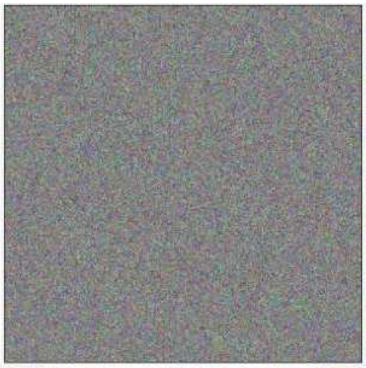

[1]: 'image_c1. jpg' (640x427x1x3)

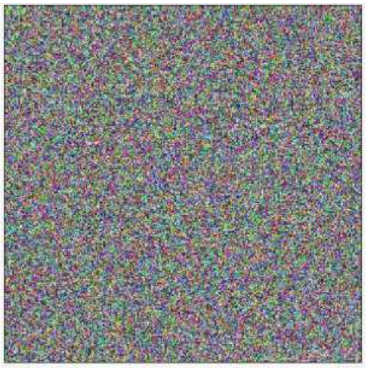

[2]: 'image_c1. jpg' (640x427x1x3)

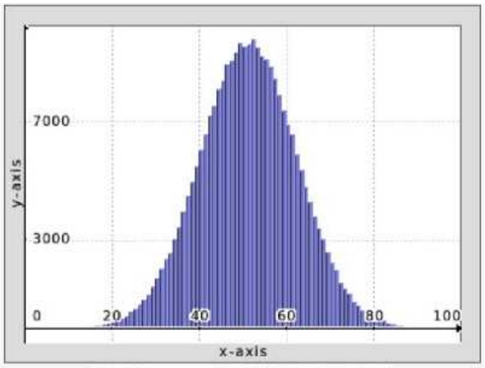

[3]: 'image_c1. jpg' (640x427x1x3)

- **Example #2**

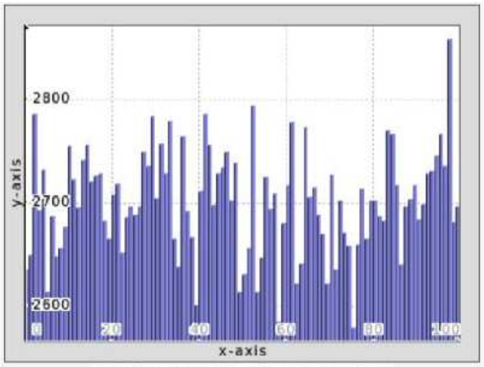
```
300,300,1,3 [0] noise[0] 20,0 noise[1] 20,1 +histogram 100
display_graph[-2,-1] 400,300,3
```


[0]: '[unnamed]' (300x300x1x3)


[1]: '[unnamed]_c1' (300x300x1x3)



[2]: '[unnamed]_c1' (400x300x1x3)



[3]: '[unnamed]_c2' (400x300x1x3)

# noise_hurl

## Arguments:

- `_amplitude>=0`

## Description:

Add hurl noise to selected images.

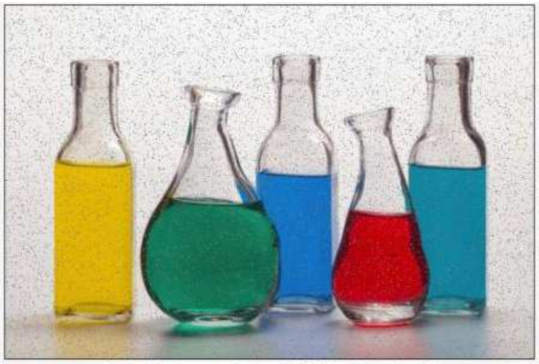
## Default values:

`amplitude=10`.

## Example of use:

```
image.jpg +noise_hurl ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# noise_perlin

## Arguments:

- `_scale_x[%]>0,_scale_y[%]>0,_scale_z[%]>0,_seed_x,_seed_y,_seed_z`

## Description:

Render 2D or 3D Perlin noise on selected images, from specified coordinates.

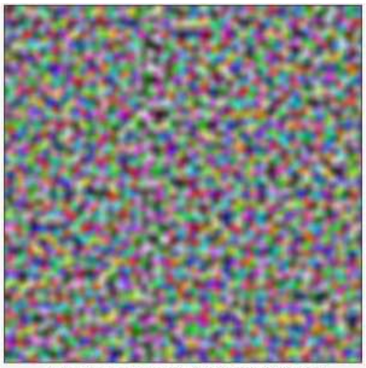
The Perlin noise is a specific type of smooth noise,
described here : **https://en.wikipedia.org/wiki/Perlin_noise**.

## Default values:

`scale_x=scale_y=scale_z=16` and `seed_x=seed_y=seed_z=0`.

## Example of use:

```
500,500,1,3 noise_perlin ,
```



[0]: '[unnamed]' (500x500x1x3)

# noise_poissondisk

## Arguments:

- `_radius[%]>0,_max_sample_attempts>0,_p_norm>0`

## Description:

Add poisson disk sampling noise to selected images.

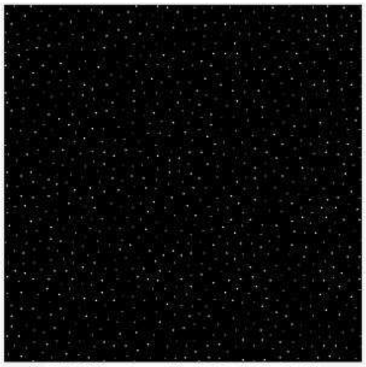
Implements the algorithm from the article "Fast Poisson Disk Sampling in Arbitrary Dimensions", by Robert Bridson (SIGGRAPH'2007).

## Default values:

`radius=8`, `max_sample_attempts=30` and `p_norm=2`.

## Example of use:

```
300,300 noise_poissondisk 8
```



[0]: '[unnamed]' (300x300x1x1)

# norm1

### No arguments

## Description:

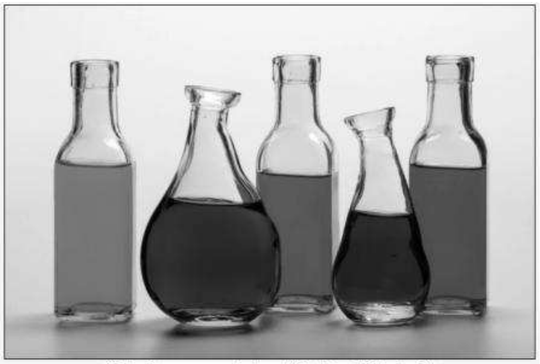
Compute the pointwise L1-norm of vector-valued pixels in selected images.

This command has a **tutorial page**.

## Example of use:

```
image.jpg +norm1
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x1)

---

# norm2

### No arguments

## Description:

Compute the pointwise L2-norm (euclidean norm) of vector-valued pixels in selected images.

This command has a **tutorial page**.

## Example of use:

```
image.jpg +norm
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x1)

## normalize

Built-in command

### Arguments:

- `{ value0[%] | [image0] },{ value1[%] | [image1] },_constant_case_ratio` or
- `[image]`

### Description:

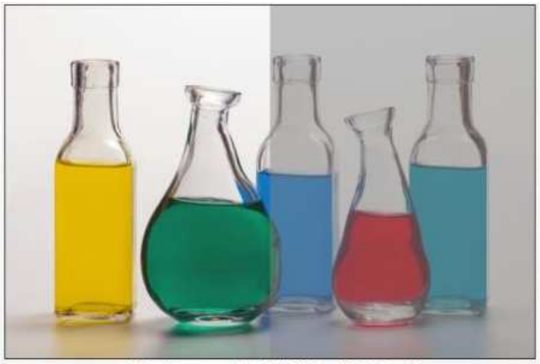
Linearly normalize values of selected images in specified range.

(*equivalent to shortcut command* `n`).

This command has a **tutorial page**.

### Example of use:

```
image.jpg split x,2 normalize[-1] 64,196 append x
```


[0]: 'image.jpg' (640x427x1x3)

## normalize3d

**No arguments**

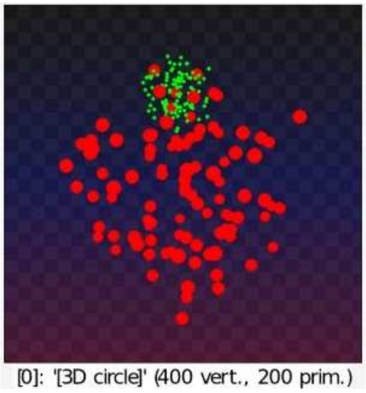
### Description:

Normalize selected 3D objects to unit size.

(*equivalent to shortcut command* `n3d`).

### Example of use:

```
repeat 100 { circle3d {u(3)},{u(3)},{u(3)},0.1 } add3d color3d[-1]
255,0,0 +normalize3d[-1] color3d[-1] 0,255,0 add3d
```



[0]: '[3D circle]' (400 vert., 200 prim.)

---

## normalize_filename

### Arguments:

- `filename`

### Description:

Return a "normalized" version of the specified filename, without spaces and capital letters.

---

## normalize_l2

**No arguments**

### Description:

Normalize selected images such that they have a unit L2 norm.

---

## normalize_local

### Arguments:

- `_amplitude>=0,_radius>0,_n_smooth>=0[%],_a_smooth>=0[%],_is_cut={ 0 | 1 },_min=0,_max=255`

## Description:

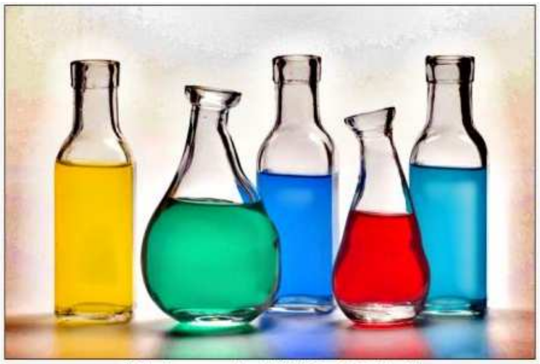
Normalize selected images locally.

## Default values:

`amplitude=3`, `radius=16`, `n_smooth=4%`, `a_smooth=2%`, `is_cut=1`, `min=0` and `max=255`.

## Example of use:

```
image.jpg normalize_local 8,10
```



[0]: 'image.jpg' (640x427x1x3)

---

# normalize_sum

### No arguments

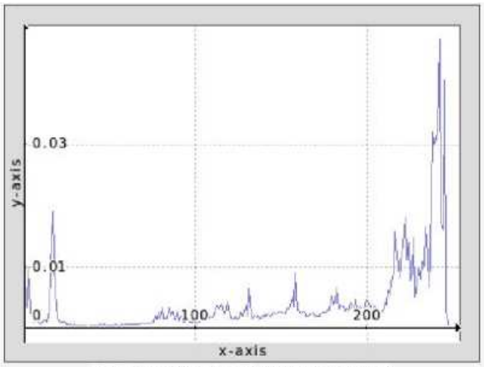
## Description:

Normalize selected images such that they have a unit sum.

## Example of use:

```
image.jpg +histogram 256 normalize_sum[-1] display_graph[-1] 400,300
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (400x300x1x3)

---

# normalized_cross_correlation

## Arguments:

- `[mask]`

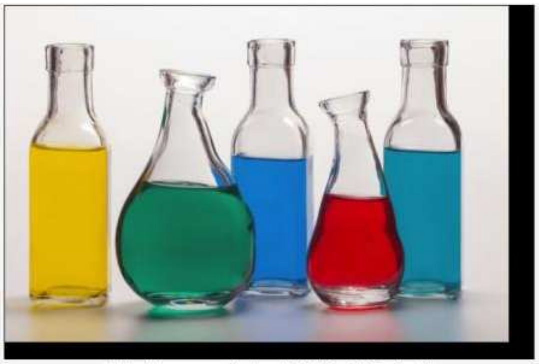
## Description:

Compute normalized cross-correlation of selected images with specified mask.

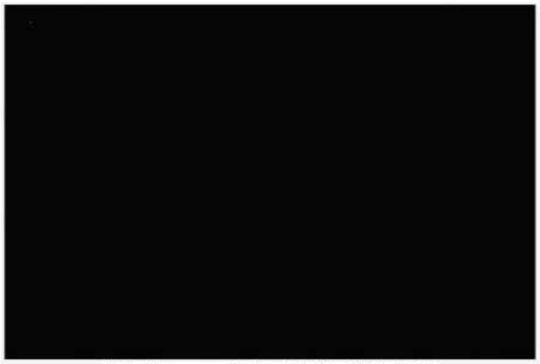
## Example of use:

```
image.jpg +shift -30,-20 +normalized_cross_correlation[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x1)

---

# normp

## Arguments:

- `p>=0`

## Description:

Compute the pointwise Lp-norm norm of vector-valued pixels in selected images.

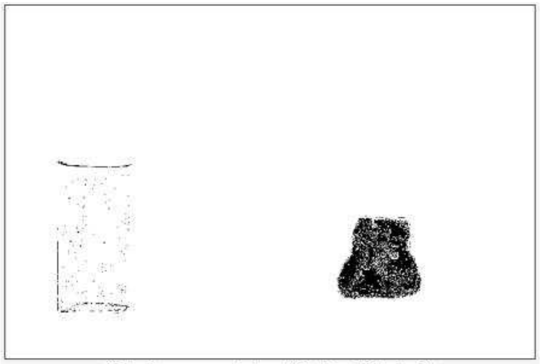
## Default values:

`p=2`.

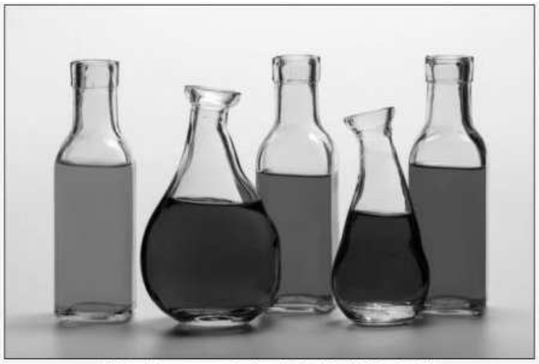
## Example of use:

```
image.jpg +normp[0] 0 +normp[0] 1 +normp[0] 2 +normp[0] inf
```

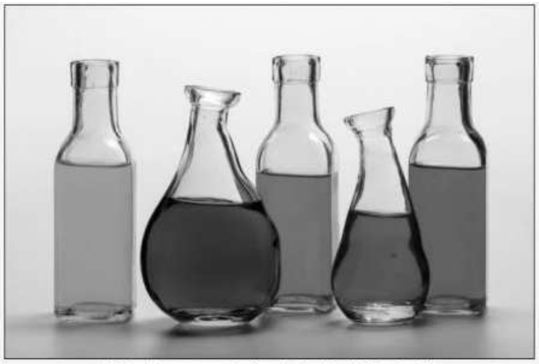
[0]: 'image.jpg' (640x427x1x3)



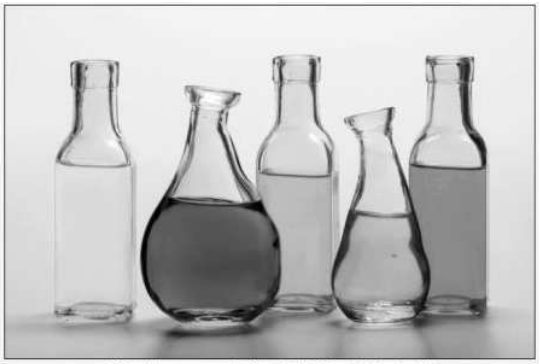
[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c1.jpg' (640x427x1x1)



[3]: 'image_c1.jpg' (640x427x1x1)



[4]: 'image_c1.jpg' (640x427x1x1)

# not

### No arguments

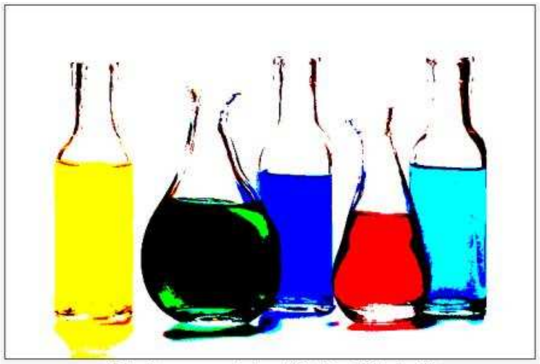
## Description:

Apply boolean not operation on selected images.

## Example of use:

```
image.jpg +ge 50% +not[-1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c2.jpg' (640x427x1x3)

# object3d

Built-in command

## Arguments:

- `[object3d],_x[%],_y[%],_z,_opacity,_rendering_mode,_is_double_sided={ 0 | 1 },_is_zbuffer={ 0 | 1 },_focale,_light_x,_light_y,_light_z,_specular_lightness,_specular_shininess`

## Description:

Draw specified 3D object on selected images.

(*equivalent to shortcut command* `j3d`).

`rendering_mode` can be ***{ 0:dots | 1:wireframe | 2:flat | 3:flat-shaded | 4:gouraud-shaded | 5:phong-shaded }***.

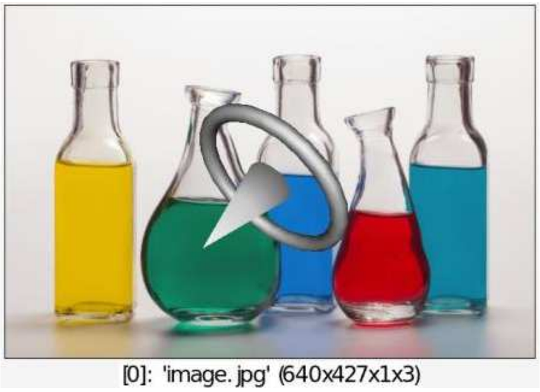
## Default values:

`x=y=z=0`, `opacity=1` and `is_zbuffer=1`. All other arguments take their default values

from the 3D environment variables.

## Example of use:

```
image.jpg torus3d 100,10 cone3d 30,-120 add3d[-2,-1] rotate3d.
1,1,0,60 object3d[0] [-1],50%,50% keep[0]
```



[0]: 'image.jpg' (640x427x1x3)

# oct

## Arguments:

- `octal_int1,...`

## Description:

Print specified octal integers into their binary, decimal, hexadecimal and string representations.

# oct2dec

## Arguments:

- `octal_int1,...`

## Description:

Convert specified octal integers into their decimal representations.

# oklab2rgb

### No arguments

## Description:

Convert color representation of selected images from OKlab to RGB.

(see colorspace definition at: **https://bottosson.github.io/posts/oklab/** ).

## See also:

**rgb2oklab** .

# old_photo

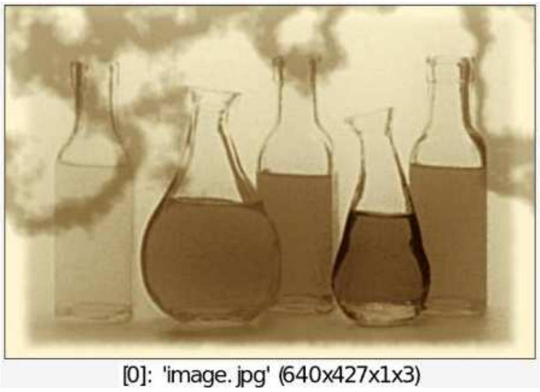
### No arguments

## Description:

Apply old photo effect on selected images.

## Example of use:

```
image.jpg old_photo
```



[0]: 'image.jpg' (640x427x1x3)

# oneminus

### No arguments

## Description:

For each selected image, compute one minus image.

## Example of use:

```
image.jpg normalize 0,1 +oneminus
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

## onfail

Built-in command

**No arguments**

### Description:

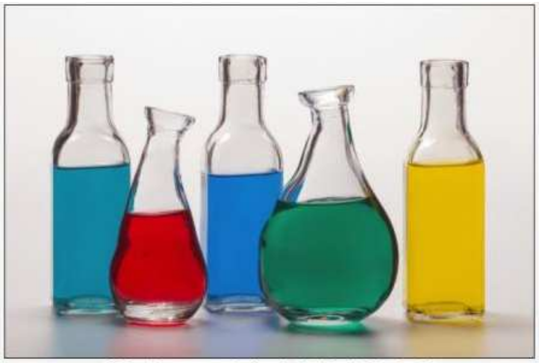
Execute following commands when an error is encountered in the body of the `local...done` block.

The status value is set with the corresponding error message.

### Example of use:

```
image.jpg +local blur -3 onfail mirror x done
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

## opacity3d

### Arguments:

- `opacity`

### Description:

Set opacity of selected 3D objects.

(*equivalent to shortcut command* `o3d`).

## Example of use:

```
torus3d 100,10 double3d 0 repeat 7 { +rotate3d[-1] 1,0,0,20
opacity3d[-1] {u} } add3d
```

[0]: '[3D torus]' (2304 vert., 2304 prim.)

---

# opening

## Arguments:

- `size>=0` or
- `size_x>=0,size_y>=0,_size_z>=0` or
- `[kernel],_boundary_conditions,_is_real={ 0:binary-mode | 1:real-mode }`

## Description:

Apply morphological opening to selected images.

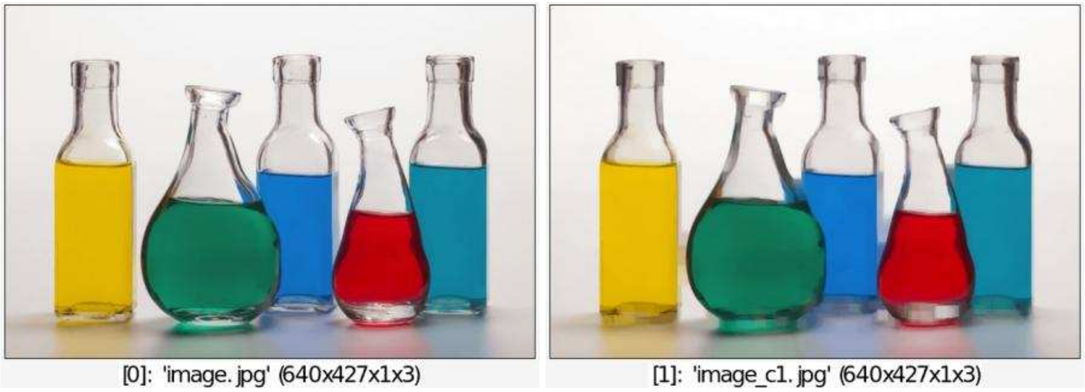
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

## Default values:

`size_z=1`, `boundary_conditions=1` and `is_real=0`.

## Example of use:

```
image.jpg +opening 10
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

---

# opening_circ

## Arguments:

- `_size>=0,_is_real={ 0 | 1 }`

## Description:

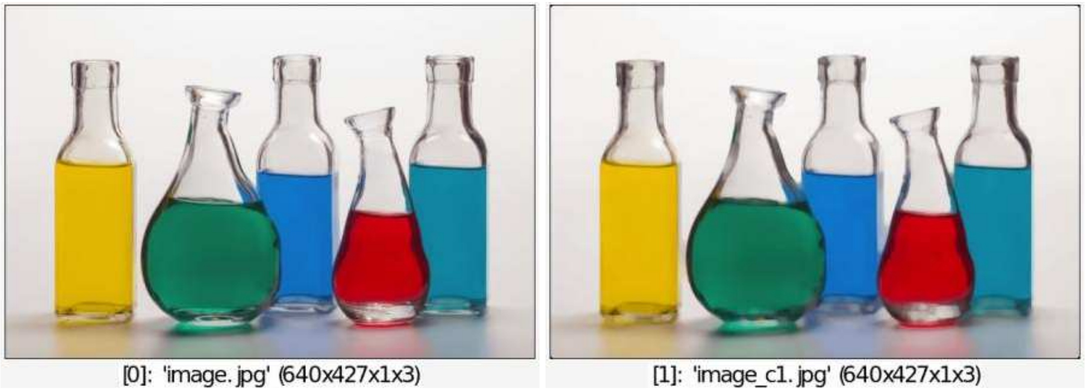
Apply circular opening of selected images by specified size.

## Default values:

`boundary_conditions=1` and `is_real=0`.

## Example of use:

```
image.jpg +opening_circ 7
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

---

# or

**Built-in command**

## Arguments:

- `value[%]` or
- `[image]` or

- `'formula'` or
- `(no arg)`

## Description:

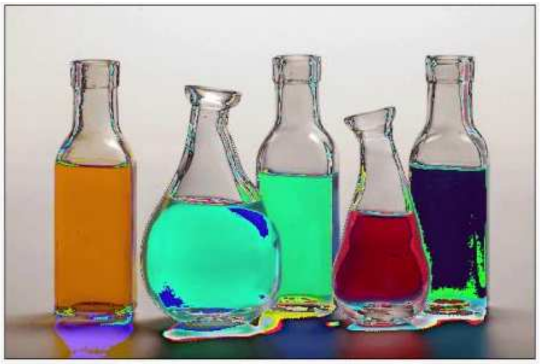
Compute the bitwise OR of selected images with specified value, image or mathematical expression, or compute the pointwise sequential bitwise OR of selected images.

(*equivalent to shortcut command* `|`).

## Examples of use:

### • Example #1

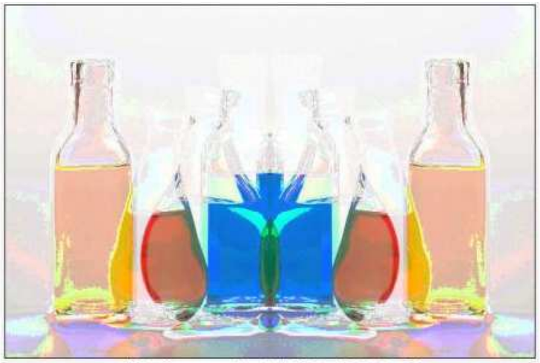
```
image.jpg or 128
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +mirror x or
```



[0]: 'image.jpg' (640x427x1x3)

---

# orientation

### No arguments

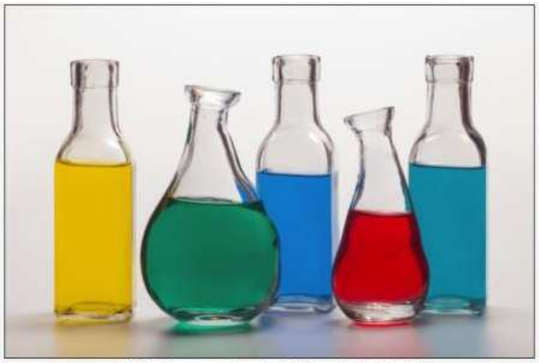
## Description:

Compute the pointwise orientation of vector-valued pixels in selected images.

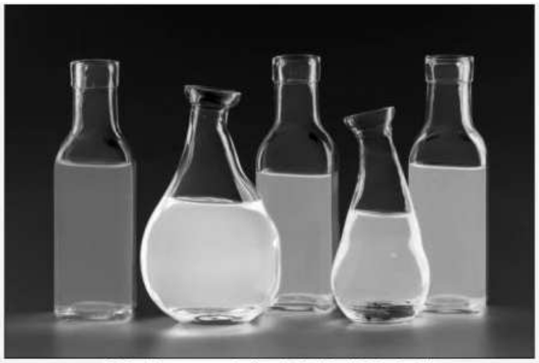
This command has a **tutorial page**.

### Example of use:

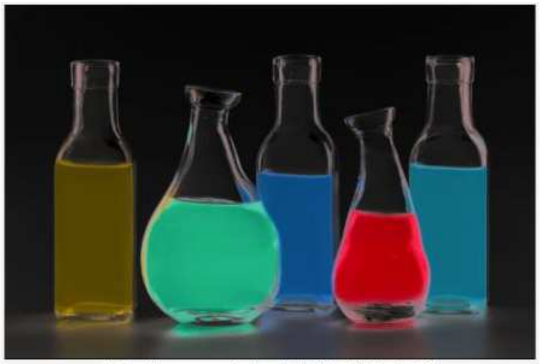
```
image.jpg +orientation +norm[-2] negate[-1] mul[-2] [-1] reverse[-2,
-1]
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

## orthogonalize

### Arguments:

- `_mode = { 0:orthogonalize | 1:orthonormalize }`

### Description:

Orthogonalize or orthonormalize selected matrices, using Modified Gram-Schmidt process.

### Default values:

`mode=0`.

## otsu

### Arguments:

- `_nb_levels>0`

### Description:

Hard-threshold selected images using Otsu's method.

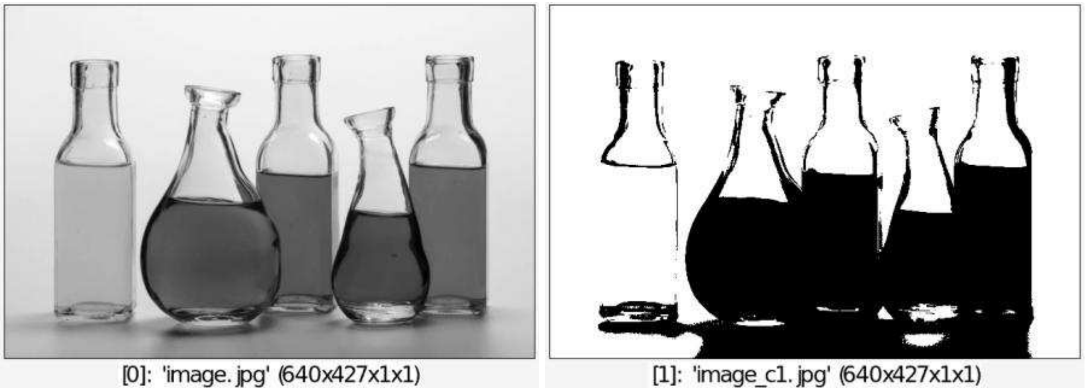
The computed thresholds are returned as a list of values in the status.

### Default values:

`nb_levels=256`.

### Example of use:

```
image.jpg luminance +otsu ,
```



[0]: 'image.jpg' (640x427x1x1)

[1]: 'image_c1.jpg' (640x427x1x1)

## output

**Built-in command**

### Arguments:

- `[type:]filename,_format_options`

### Description:

Output selected images as one or several numbered file(s).

(*equivalent to shortcut command* `o`).

### Default values:

'format_options'=(undefined).

## output_565

## Arguments:

- `"filename",reverse_endianness={ 0:false | 1:true }`

## Description:

Output selected images as raw RGB-565 files.

## Default values:

`reverse_endianness=0`.

---

# output_cube

## Arguments:

- `"filename"`

## Description:

Output selected CLUTs as a .cube file (Adobe CLUT format).

---

# output_flo

## Arguments:

- `"filename"`

## Description:

Output selected optical flow as a .flo file (vision.middlebury.edu file format).

---

# output_ggr

## Arguments:

- `filename,_gradient_name`

## Description:

Output selected images as .ggr gradient files (GIMP).

If no gradient name is specified, it is deduced from the filename.

---

# output_gmz

## Arguments:

- `filename,_datatype`

## Description:

Output selected images as .gmz files (G'MIC native file format).

`datatype` can be ***{ bool | uint8 | int8 | uint16 | int16 | uint32 | int32 | uint64 | int64 | float32 | float64 }***.

---

# output_obj

## Arguments:

- `filename,_save_materials={ 0:no | 1:yes }`

## Description:

Output selected 3D meshes as Wavefront 3D object files.

Set `save_materials` to `1` to produce a corresponding material file (`.mtl`) and eventually texture files.
Beware, the export to `.obj` files may be quite slow for large 3D objects.

## Default values:

`save_materials=1`.

---

# output_text

## Arguments:

- `filename`

## Description:

Output selected images as text-data filenames.

(*equivalent to shortcut command* `ot`).

---

# outputn

## Arguments:

- `filename,_index`

## Description:

Output selected images as automatically numbered filenames in repeat...done loops.

(*equivalent to shortcut command* `on`).

---

# outputp

## Arguments:

- `prefix`

## Description:

Output selected images as prefixed versions of their original filenames.

(*equivalent to shortcut command* `op`).

## Default values:

`prefix=_`.

---

# outputw

### No arguments

## Description:

Output selected images by overwriting their original location.

(*equivalent to shortcut command* `ow`).

---

# outputx

## Arguments:

- `extension1,_extension2,_...,_extensionN,_output_at_same_location={ 0 | 1 }`

## Description:

Output selected images with same base filenames but for N different extensions.

(*equivalent to shortcut command* `ox`).

## Default values:

`output_at_same_location=0`.

---

# pack

## Arguments:

- `is_ratio_constraint={ 0 | 1 },_sort_criterion`

## Description:

Pack selected images into a single image.

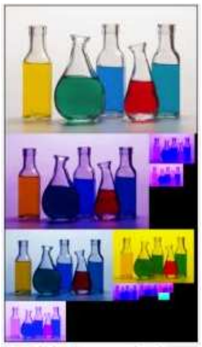
The returned status contains the list of new (x,y) offsets for each input image.
Parameter `is_ratio_constraint` tells if the resulting image must tend to a square image.

## Default values:

`is_ratio_constraint=0` and `sort_criterion=max(w,h)`.

## Example of use:

```
image.jpg repeat 10 +rescale2d[-1] 75% balance_gamma[-1] ${-rgb} done
pack 0
```



[0]: 'image.jpg'
(640x1122x1x3)

---

# pack_sprites

## Arguments:

- `_nb_scales>=0,0<=_min_scale<=100,_allow_rotation={ 0:0 deg. | 1:180 deg.`

`| 2:90 deg. | 3:any },_spacing,_precision>=0,max_iterations>=0`

## Description:

Try to randomly pack as many sprites as possible onto the `empty` areas of an image.

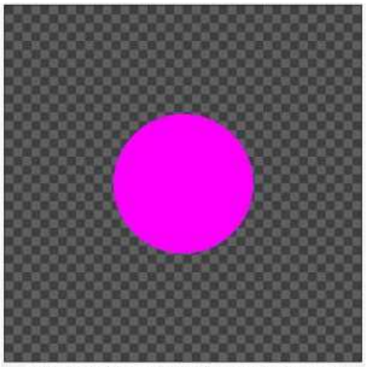
Sprites can be eventually rotated and scaled during the packing process.
First selected image is the canvas that will be filled with the sprites.
Its last channel must be a binary mask whose zero values represent potential locations for drawing the sprites.
All other selected images represent the sprites considered for packing.
Their last channel must be a binary mask that represents the sprite shape (i.e. a 8-connected component).
The order of sprite packing follows the order of specified sprites in the image list.
Sprite packing is done on random locations and iteratively with decreasing scales.
`nb_scales` sets the number of decreasing scales considered for all specified sprites to be packed.
`min_scale` (in %) sets the minimal size considered for packing (specified as a percentage of the original sprite size).
`spacing` can be positive or negative.
`precision` tells about the desired number of failed trials before ending the filling process.

## Default values:

`nb_scales=5`, `min_scale=25`, `allow_rotation=3`, `spacing=1`, `precision=7` and `max_iterations=256`.

## Example of use:

```
512,512,1,3,"min(255,y*c/2)" 100%,100% circle 50%,50%,100,1,255
append c image.jpg rescale2d[-1] ,24 to_rgba pack_sprites 3,25
```



[0]: '[min(255,y*c/2)]' (512x512x1x4)

# padint

## Arguments:

- `number,_size>0`

## Description:

Return a integer with `size` digits (eventually left-padded with `0`).

# palette

## Arguments:

- `palette_name | palette_number`

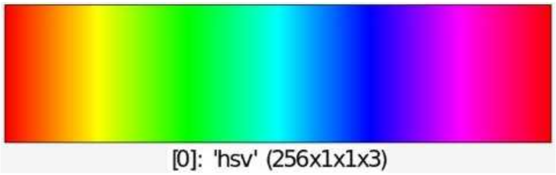
## Description:

Input specified color palette at the end of the image list.

`palette_name` can be *{ default | hsv | lines | hot | cool | jet | flag | cube | rainbow | parula | spring | summer | autumn | winter | bone | copper | pink | vga | algae | amp | balance | curl | deep | delta | dense | diff | gray | haline | ice | matter | oxy | phase | rain | solar | speed | tarn | tempo | thermal | topo | turbid | aurora | hocuspocus | srb2 | uzebox | amiga7800 | amiga7800mess | fornaxvoid1 }*

## Example of use:

```
palette hsv
```


[0]: 'hsv' (256x1x1x3)

# parallel

**Built-in command**

## Arguments:

- `_wait_threads,"command1","command2",...`

## Description:

Execute specified commands in parallel, each in a different thread.

Parallel threads share the list of images.
`wait_threads` can be *{ 0:when current environment ends | 1:immediately }*.

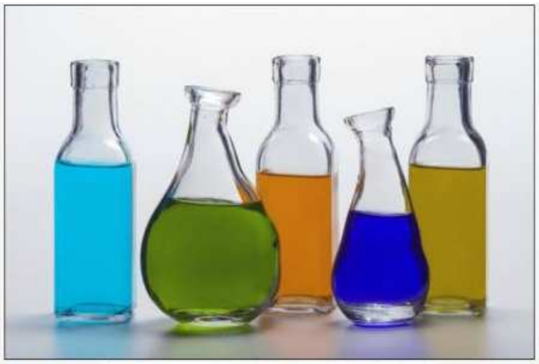
## Default values:

`wait_threads=1`.

## Example of use:

```
image.jpg [0] parallel "blur[0] 3","mirror[1] c"
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# parametric3d

## Arguments:

- `_x(a,b),_y(a,b),_z(a,b),_amin,_amax,_bmin,_bmax,_res_a>0,_res_b>0,_res_x>0,_r`

## Description:

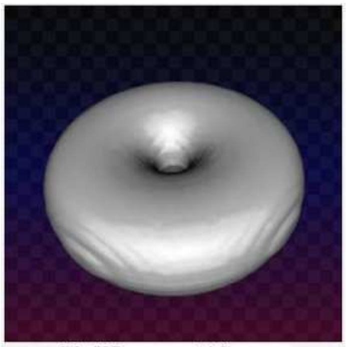
Input 3D object from specified parametric surface `(a,b) → (x(a,b),y(a,b),z(a,b))`.

## Default values:

`x=(2+cos(b))*sin(a)`, `y=(2+cos(b))*cos(a)`, `c=sin(b)`, `amin=-pi`, `amax=pi`, `bmin=-pi`, `bmax=pi`, `res_a=512`, `res_b=res_a`, `res_x=64`, `res_y=res_x`, `res_z=res_y`, `smoothness=2%` and `isovalue=10%`.

## Example of use:

```
parametric3d ,
```



[0]: '[3D parametric]'
(12020 vert., 24040 prim.)

# parse_cli

## Arguments:

- `_output_mode,_{ * | command_name }`

## Description:

Parse definition of `@cli`-documented commands and output info about them in specified output mode.

`output_mode` can be ***{ ascii | bashcompletion | html | images | print }***.

## Default values:

`output_mode=print` and `command_name=*`.

---

# parse_gmd

### No arguments

## Description:

Parse and tokenize selected images, viewed as text strings formatted with the G'MIC markdown syntax.

---

# parse_gui

## Arguments:

- `_outputmode,_{ * | filter_name}`

## Description:

Parse selected filter definitions and generate info about filters in selected output mode.

`outputmode` can be ***{ gmicol | images | json | list | print | strings | update | zart }***.

It is possible to define a custom output mode, by implementing the following commands
(`outputmode` must be replaced by the name of the custom user output mode):
. `parse_gui_outputmode` : A command that outputs the parsing information with a custom format.
. `parse_gui_parseparams_outputmode` (optional): A simple command that returns 0 or 1. It tells the parser whether parameters of matching filter must be analyzed (slower) or not.
. `parse_gui_trigger_outputmode` (optional): A command that is called by the parser just before parsing the set of each matching filters.

Here is the list of global variables set by the parser, accessible in command `parse_gui_outputmode`:

`$_nb_filters` : Number of matching filters.
`$_nongui` (stored as an image): All merged lines in the file that do not correspond to `#@gui` lines.

For each filter `#F` ( `F` in range `[0,$_nb_filters-1]`):

- `$_fF_name` : Filter name.
- `$_fF_path` : Full path.
- `$_fF_locale` : Filter locale (empty, if not specified).
- `$_fF_command` : Filter command.
- `$_fF_command_preview` : Filter preview command (empty, if not specified).
- `$_fF_zoom_factor` : Default zoom factor (empty, if not specified).
- `$_fF_preview_accuracy` : Preview accuracy (can be ***`{ 0:does not support zoom in/out | 1:support zoom in/out | 2:pixel-perfect }`***).
- `$_fF_input_mode` : Default preferred input mode (empty, if not specified).
- `$_fF_hide` : Path of filter hid by current filter (for localized filters, empty if not specified).
- `$_fF_nb_params` : Number of parameters.

For each parameter `#P` of the filter #F ( `P` in range `[0,$_fF_nb_params-1]`):

- `$_fF_pP_name` : Parameter name.
- `$_fF_pP_type` : Parameter type.
- `$_fF_pP_responsivity` : Parameter responsivity (can be ***`{ 0 | 1 }`***).
- `$_fF_pP_visibility` : Parameter visibility.
- `$_fF_pP_propagation` : Propagation of the parameter visibility.
- `$_fF_pP_nb_args` : Number of parameter arguments.

For each argument `#A` of the parameter #P ( `A` in range `[0,$_fF_pP_nb_args-1]`):

- `$_fF_pP_aA` : Argument value

Default parameters: `filter_name=*` and `output_format=print`.

---

# pass

**Built-in command**

## Arguments:

- `_shared_state={ -1:status only | 0:non-shared (copy) | 1:shared | 2:adaptive }`

## Description:

Insert images from parent context of a custom command or a local environment.

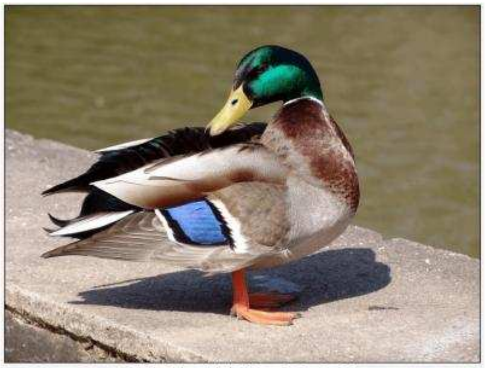
Command selection (if any) stands for a selection of images in the parent context.
By default (adaptive shared state), selected images are inserted in a shared state if they do not belong
to the context (selection) of the current custom command or local environment as well.
Typical use of command `pass` concerns the design of custom commands that take images as arguments.
This commands return the list of corresponding indices in the status.

## Default values:

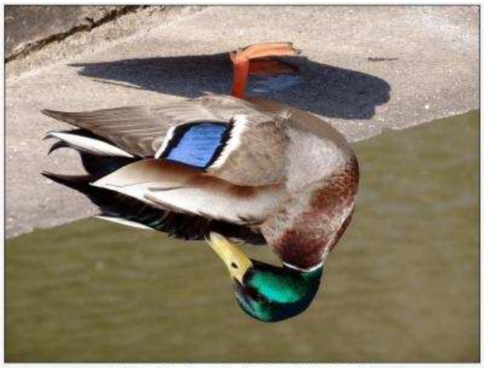
`shared_state=2`.

## Example of use:

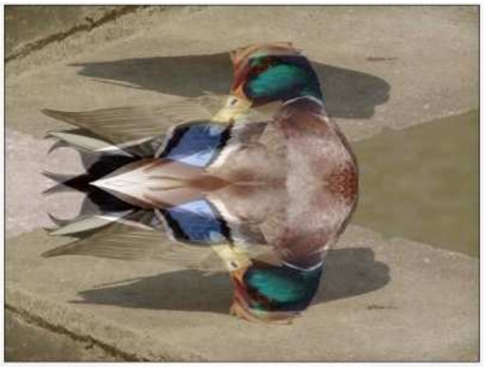
```
command "average : pass$""1 add[^-1] [-1] remove[-1] div 2" sample ?
+mirror y +average[0] [1]
```



[0]: 'duck' (640x480x1x3)



[1]: 'duck_c1' (640x480x1x3)



[2]: 'duck_c1' (640x480x1x3)

# patches

## Arguments:

- `patch_width>0,patch_height>0,patch_depth>0,x0,y0,z0,_x1,_y1,_z1,...,_xN,_yN,_`

## Description:

Extract N+1 patches from selected images, centered at specified locations.

## Example of use:

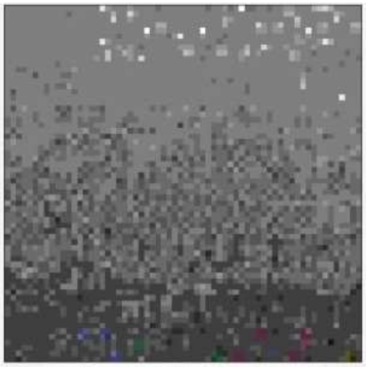
```
image.jpg +patches 64,64,1,153,124,0,184,240,0,217,126,0,275,38,0
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (64x64x1x3)

[2]: 'image_c2.jpg' (64x64x1x3)

[3]: 'image_c2.jpg' (64x64x1x3)



[4]: 'image_c3.jpg' (64x64x1x3)

# patches2img

## Arguments:

- `width>0,height>0,_overlap[%]>0,_overlap_std[%]`

## Description:

Recompose 2D images from their selected patch representations.

`overlap` must be in range `[0,patch_size-1]` where `patch_size` is the width/height of the selected image.
`overlap_std` is the standard deviation of the gaussian weights used for reconstructing overlapping patches.
If `overlap_std` is set to `-1`, uniform weights are used rather than gaussian.

## Default values:

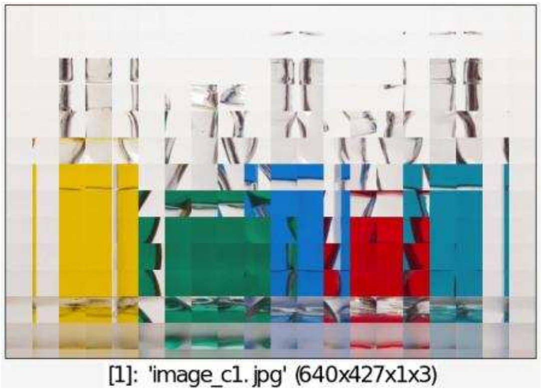
`overlap=0` and `overlap_std=-1`.

## See also:

**img2patches**.

## Example of use:

```
image.jpg +img2patches 32,0,3 mirror[-1] xy patches2img[-1] {0,[w,h]}
```



# path_cache

### No arguments

## Description:

Return a path to store G'MIC data files for one user (whose value is OS-dependent).

# path_current

### No arguments

## Description:

Return current folder from where G'MIC has been run.

# path_gimp

### No arguments

## Description:

Return a path to store GIMP configuration files for one user (whose value is OS-dependent).

# path_tmp

### No arguments

## Description:

Return a path to store temporary files (whose value is OS-dependent).

# pca_patch3d

## Arguments:

- `_patch_size>0,_M>0,_N>0,_normalize_input={ 0 | 1 },_normalize_output={ 0 | 1 },_lambda_xy`

## Description:

Get 3D patch-pca representation of selected images.

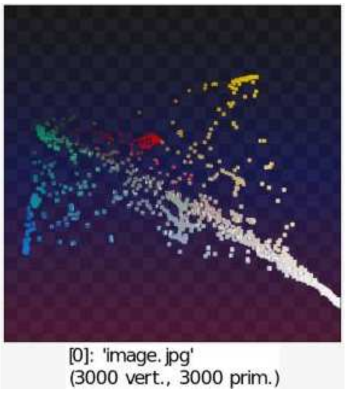
The 3D patch-pca is estimated from M patches on the input image, and displayed as a cloud of N 3D points.

## Default values:

`patch_size=7`, `M=1000`, `N=3000`, `normalize_input=1`, `normalize_output=0`, and `lambda_xy=0`.

## Example of use:

```
image.jpg pca_patch3d 7
```



[0]: 'image.jpg'
(3000 vert., 3000 prim.)

# pde_flow

## Arguments:

- `_nb_iter>=0,_dt,_velocity_command,_keep_sequence={ 0 | 1 }`

## Description:

Apply iterations of a generic PDE flow on selected images.

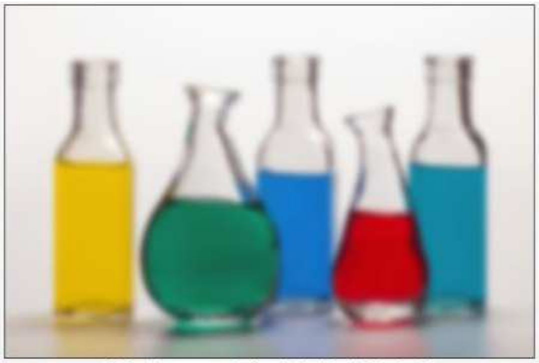
## Default values:

`nb_iter=10`, `dt=30`, `velocity_command=laplacian` and `keep_sequence=0`.

## Example of use:

```
image.jpg +pde_flow 20
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

# pencilbw

## Arguments:

- `_size>=0,_amplitude>=0`

## Description:

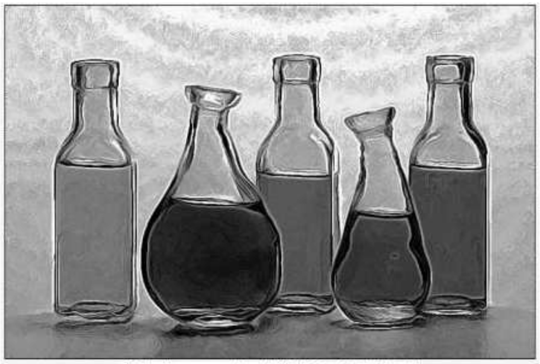
Apply B&W pencil effect on selected images.

## Default values:

`size=0.3` and `amplitude=60`.

## Example of use:

```
image.jpg pencilbw ,
```


[0]: 'image.jpg' (640x427x1x1)

# percentile

## Arguments:

- `[mask],0<=_min_percentile[%]<=100,0<=_max_percentile[%]<=100.`

## Description:

Apply percentile averaging filter to selected images.

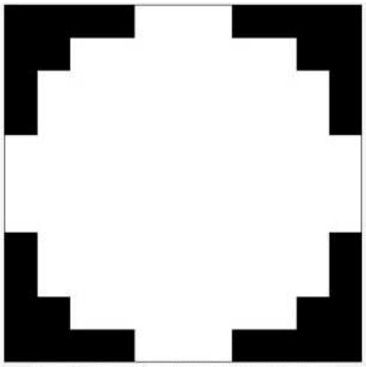
## Default values:

`min_percentile=0` and `max_percentile=100`.

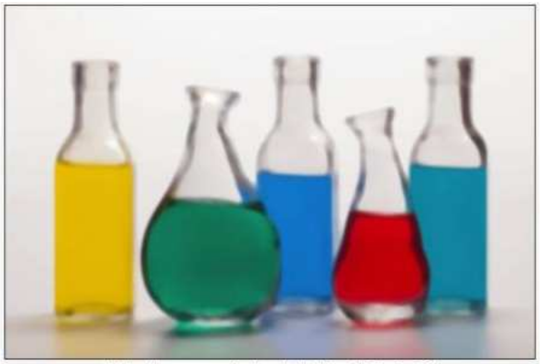
## Example of use:

```
image.jpg shape_circle 11,11 +percentile[0] [1],25,75
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[2D circle shape]' (11x11x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

# periodize_poisson

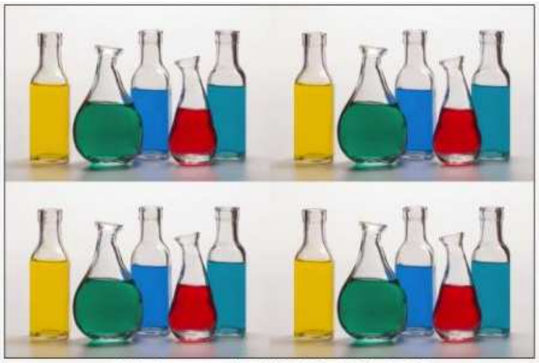
**No arguments**

## Description:

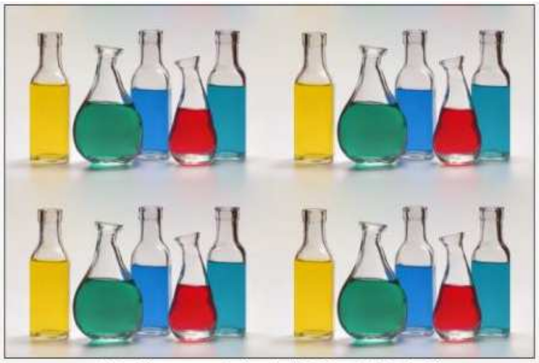
Periodize selected images using a Poisson solver in Fourier space.

## Example of use:

```
image.jpg +periodize_poisson array 2,2,2
```



[0]: 'image.jpg' (1280x854x1x3)



[1]: 'image_c1.jpg' (1280x854x1x3)

---

# permute

Built-in command

## Arguments:

- `permutation_string`

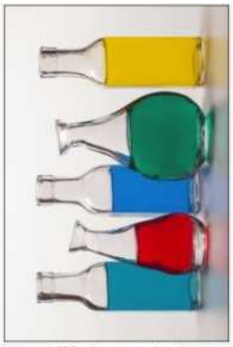
## Description:

Permute selected image axes by specified permutation.

`permutation` is a combination of the character set ***{x|y|z|c}***,
e.g. `xycz`, `cxyz`, ...

## Example of use:

```
image.jpg permute yxzc
```



[0]: 'image.jpg'
(427x640x1x3)

---

# peronamalik_flow

## Arguments:

- `K_factor>0,_nb_iter>=0,_dt,_keep_sequence={ 0 | 1 }`

## Description:

Apply iterations of the Perona-Malik flow on selected images.

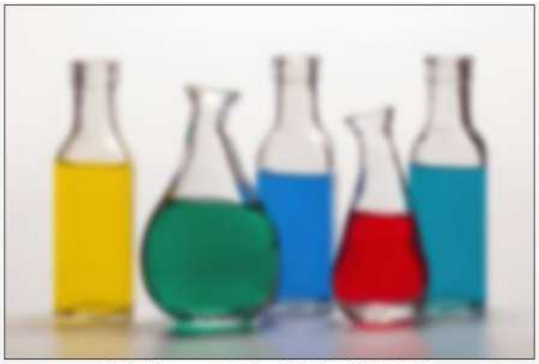
## Default values:

`K_factor=20`, `nb_iter=5`, `dt=5` and `keep_sequence=0`.

## Example of use:

```
image.jpg +heat_flow 20
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# phase_correlation

## Arguments:

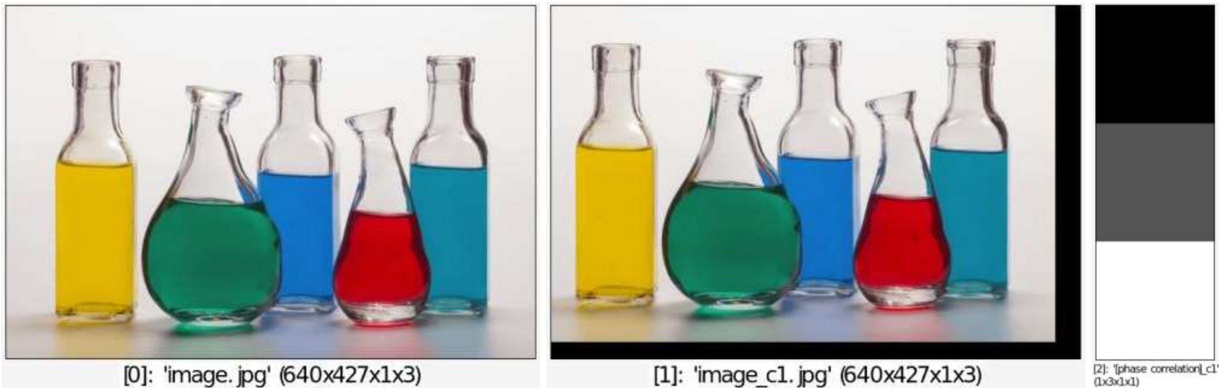
- `[destination]`

## Description:

Estimate translation vector between selected source images and specified destination.

## Example of use:

```
image.jpg +shift -30,-20 +phase_correlation[0] [1] unroll[-1] y
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)
[2]: '[phase correlation]_c1' (1x3x1x1)

# piechart

## Arguments:

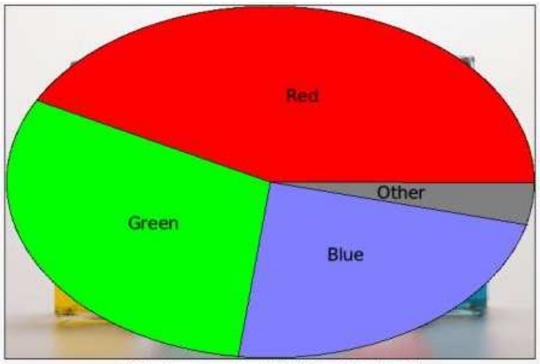
- `label_height>=0,label_R,label_G,label_B,"label1",value1,R1,G1,B1,...,"labelN"`

## Description:

Draw pie chart on selected (RGB) images.

## Example of use:

```
image.jpg piechart
25,0,0,0,"Red",55,255,0,0,"Green",40,0,255,0,"Blue",30,128,128,255,"Other",5
```



[0]: 'image.jpg' (640x427x1x3)

# pixelize

## Arguments:

- `_scale_x>0,_scale_y>0,_scale_z>0`

## Description:

Pixelize selected images with specified scales.

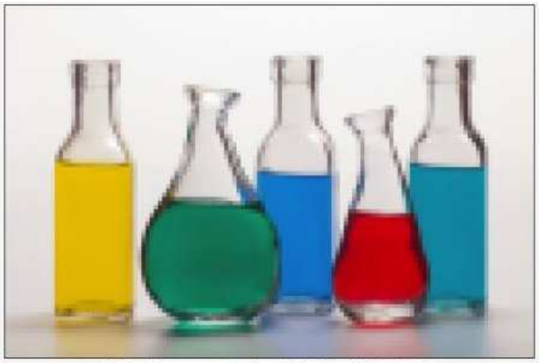
## Default values:

`scale_x=20` and `scale_y=scale_z=scale_x`.

## Example of use:

```
image.jpg +pixelize ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# pixelsort

## Arguments:

- `_ordering={ + | - },_axis={ x | y | z | xy | yx },_[sorting_criterion],_[mask]`

## Description:

Apply a 'pixel sorting' algorithm on selected images, as described in the page :

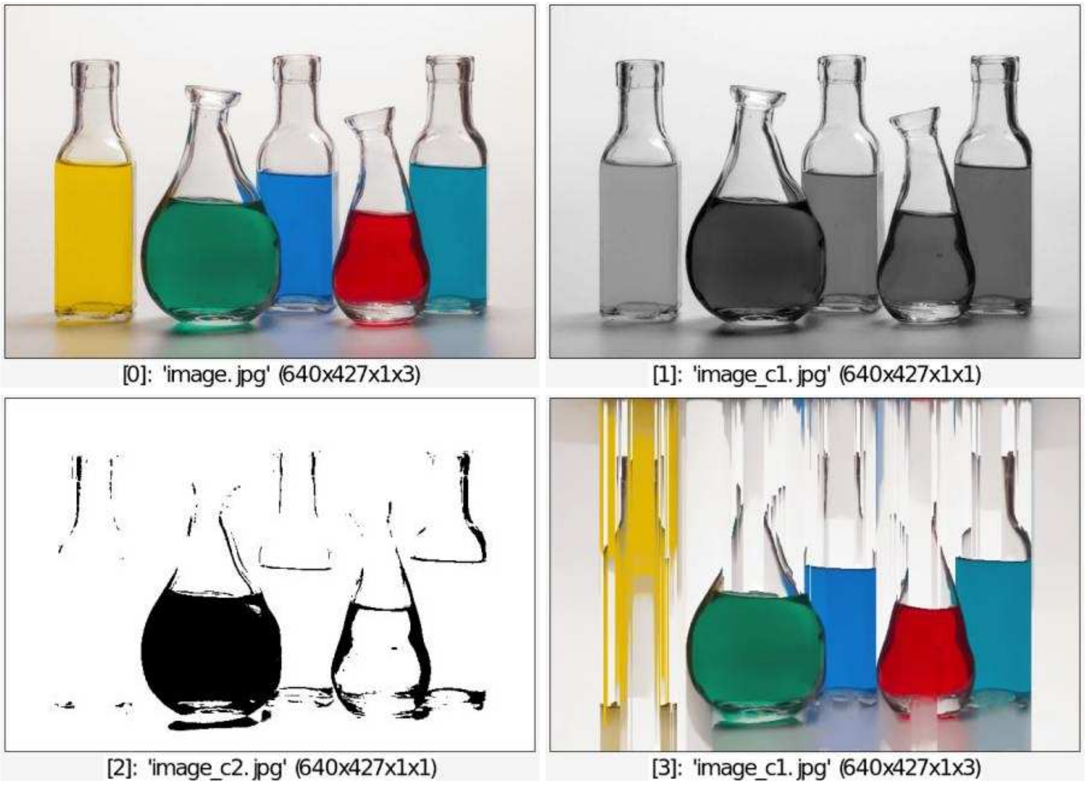
**http://satyarth.me/articles/pixel-sorting/**.

## Default values:

`ordering=+`, `axis=x` and `sorting_criterion=mask=(undefined)`.

## Example of use:

```
image.jpg +norm +ge[-1] 30% +pixelsort[0] +,y,[1],[2]
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x1)
[2]: 'image_c2.jpg' (640x427x1x1)
[3]: 'image_c1.jpg' (640x427x1x3)

# plane3d

## Arguments:

- `_size_x,_size_y,_nb_subdivisions_x>0,_nb_subdisivions_y>0`

## Description:

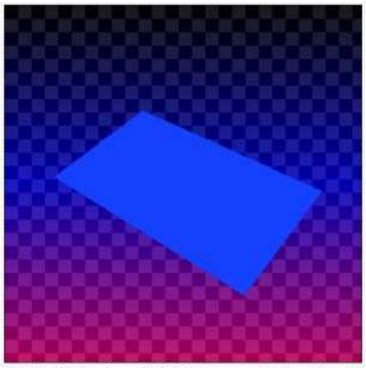
Input 3D plane at (0,0,0), with specified geometry.

## Default values:

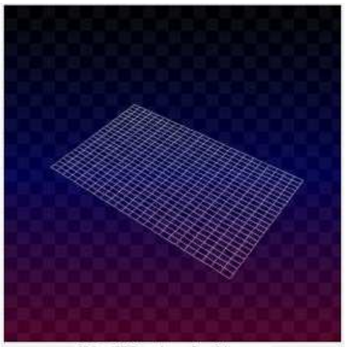
`size_x=1`, `size_y=size_x` and `nb_subdivisions_x=nb_subdivisions_y=24`.

## Example of use:

```
plane3d 50,30 +primitives3d 1 color3d[-2] ${-rgb}
```



[0]: '[3D plane]' (625 vert., 576 prim.)



[1]: '[3D plane]_c1'
(625 vert., 1200 prim.)

# plasma

Built-in command

## Arguments:

- `_alpha,_beta,_scale>=0`

## Description:

Draw a random colored plasma fractal on selected images.

This command implements the so-called `Diamond-Square` algorithm.

## Default values:

`alpha=1`, `beta=1` and `scale=8`.

This command has a **tutorial page**.

## Example of use:

```
400,400,1,3 plasma
```

[0]: '[unnamed]' (400x400x1x3)

# plot

Built-in command

## Arguments:

- `_plot_type,_vertex_type,_xmin,_xmax,_ymin,_ymax,_exit_on_anykey={ 0 | 1 }` or
- `'formula',_resolution>=0,_plot_type,_vertex_type,_xmin,xmax,_ymin,_ymax,_exit 0 | 1 }`

## Description:

Display selected images or formula in an interactive viewer (use the instant display window [0] if opened).

`plot_type` can be ***{ 0:none | 1:lines | 2:splines | 3:bar }***.
`vertex_type` can be ***{ 0:none | 1:points | 2,3:crosses | 4,5:circles | 6,7:squares }***.
`xmin`, `xmax`, `ymin`, `ymax` set the coordinates of the displayed xy-axes.

## Default values:

`plot_type=1`, `vertex_type=1`, 'xmin=xmax=ymin=ymax=0 (auto)' and `exit_on_anykey=0`.

---

# plot2value

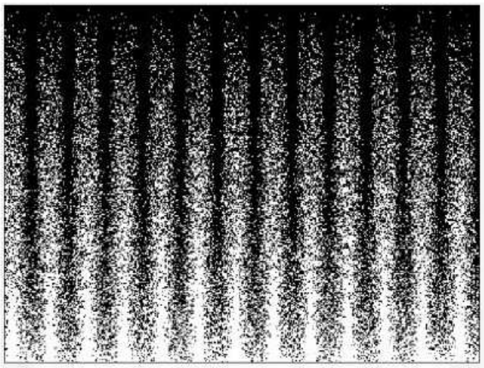
### No arguments

## Description:

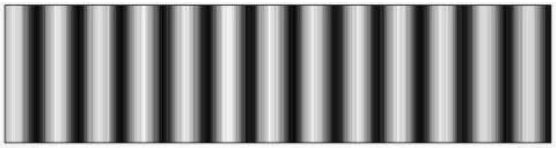
Retrieve values from selected 2D graph plots.

## Example of use:

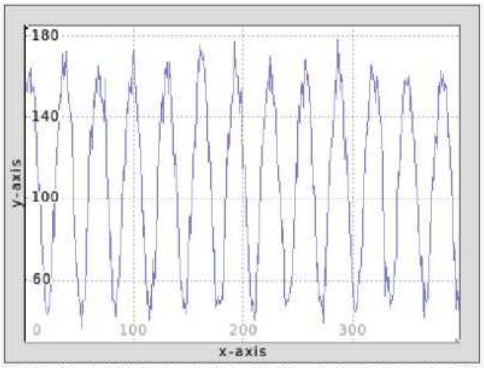
```
400,300,1,1,'y>300*abs(cos(x/10+2*u))' +plot2value +display_graph[-1]
400,300
```



[0]: '[y>300*abs(cos(x/10+2*u))]' (400x300x1x1)



[1]: '[y>300*abs(cos(x/10+2*u))]_c1' (400x1x1x1)



[2]: '[y>300*abs(cos(x/10+2*u))]_c2' (400x300x1x3)

# poincare_disk

## Arguments:

- `_size>=0,_p>2,_q>2,_angle,_tiling={ 0:triangular | 1:polygonal },_nb_max_iter>=0,_xmin,_ymin,_xmax,_ymax`

## Description:

Return a 3-channels image of a poincare disk. Output channels are `[x,y,it]`.

## Default values:

`size=1024`, `p=5`, `q=3`, `angle=0`, `tiling=0`, `nb_max_iter=20`, `xmin=ymin=-1` and `xmax=ymax=1`.

repeat 4 *{ poincare_disk 1024,{3+$>}* channels[-1] 2 mod[-1] 3 neq[-1] 2 } rescale2d 50%

# point

Built-in command

## Arguments:

- `x[%],_y[%],_z[%],_opacity,_color1,...`

## Description:

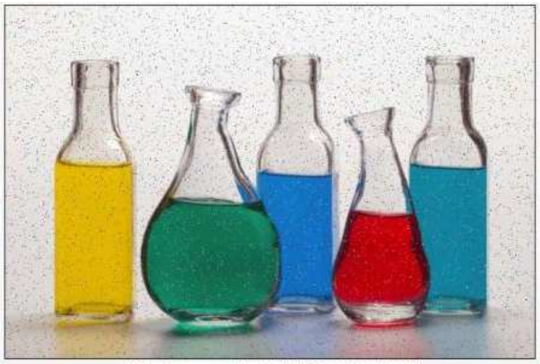
Set specified colored pixel on selected images.

## Default values:

`z=0`, `opacity=1` and `color1=0`.

## Example of use:

```
image.jpg repeat 10000 point {u(100)}%,{u(100)}%,0,1,${-rgb} done
```



[0]: 'image.jpg' (640x427x1x3)

# point3d

## Arguments:

- `x0,y0,z0`

## Description:

Input 3D point at specified coordinates.

## Example of use:

```
repeat 1000 { a:=$>*pi/500 point3d {cos(3*$a)},{sin(2*$a)},0
color3d[-1] ${-rgb} } add3d
```

[0]: '[3D point]' (1000 vert., 1000 prim.)

# pointcloud

## Arguments:

- `_type = { -X:-X-opacity | 0:binary | 1:cumulative | 2:label | 3:retrieve coordinates },_width,_height>0,_depth>0`

## Description:

Render a set of point coordinates, as a point cloud in a 1D/2D or 3D binary image

(or do the reverse, i.e. retrieve coordinates of non-zero points from a rendered point cloud).
Input point coordinates can be a NxMx1x1, Nx1x1xM or 1xNx1xM image, where `N` is the number of points,
and M the point coordinates.
If 'M'>3, the 3-to-M components sets the (M-3)-dimensional color at each point.
Parameters `width`, `height` and `depth` are related to the size of the final image :

- If set to 0, the size is automatically set along the specified axis.
- If set to N>0, the size along the specified axis is N.
- If set to N<0, the size along the specified axis is at most N.

Points with coordinates that are negative or higher than specified (`width`, `height`, `depth`) are not plotted.

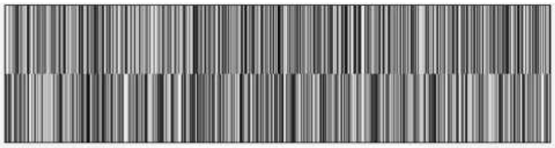
## Default values:

`type=0` and `max_width=max_height=max_depth=0`.

## Examples of use:

### • Example #1

```
3000,2 rand 0,400 +pointcloud 0 dilate[-1] 3
```

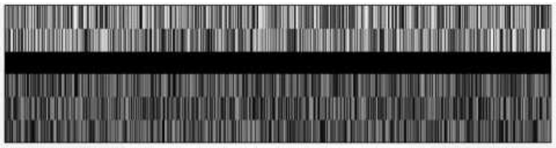


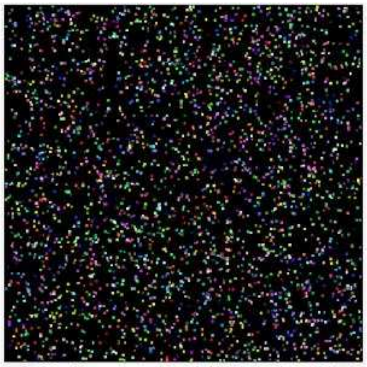
[0]: '[unnamed]' (3000x2x1x1)

[1]: '[unnamed]_c1' (401x401x1x1)

### • Example #2

```
3000,2 rand 0,400 {w} {w},3 rand[-1] 0,255 append y +pointcloud 0
dilate[-1] 3
```


[0]: '[unnamed]' (3000x6x1x1)


[1]: '[unnamed]_c1' (401x401x1x3)

# pointcloud3d

**No arguments**

## Description:

Convert selected planar or volumetric images to 3D point clouds.

## Example of use:

```
image.jpg luminance rescale2d ,100 threshold 50% mul 255 pointcloud3d
color3d[-1] 255,255,255
```

[0]: 'image.jpg'
(10975 vert., 10975 prim.)

# polar2complex

**No arguments**

## Description:

Compute polar to complex transforms of selected images.

## polar2euclidean

### Arguments:

- `_center_x[%],_center_y[%],_stretch_factor>0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

### Description:

Apply euclidean to polar transform on selected images.

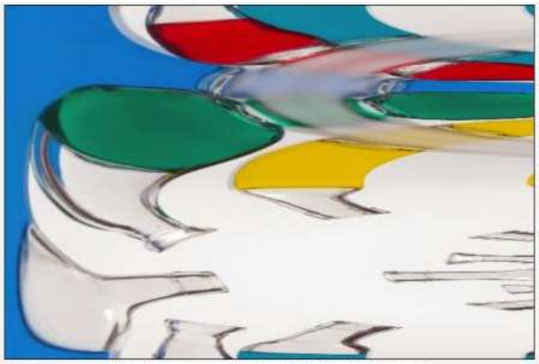
### Default values:

`center_x=center_y=50%`, `stretch_factor=1` and `boundary_conditions=3`.

### Example of use:

```
image.jpg +euclidean2polar ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

## polaroid

### Arguments:

- `_size1>=0,_size2>=0`

### Description:

Create polaroid effect in selected images.

### Default values:

`size1=10` and `size2=20`.

### Example of use:

```
image.jpg to_rgba polaroid 5,30 rotate 20 drop_shadow , drgba
```

[0]: 'image.jpg' (868x814x1x3)

## polka_dots

### Arguments:

- `diameter>=0,_density,_offset1,_offset2,_angle,_aliasing,_shading,_opacity,_cc`

### Description:

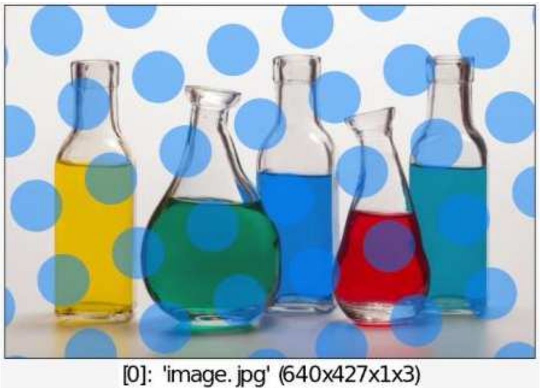
Draw dots pattern on selected images.

### Default values:

`density=20`, `offset1=offset2=50`, `angle=0`, `aliasing=10`, `shading=1`, `opacity=1` and `color=255`.

### Example of use:

```
image.jpg polka_dots 10,15,0,0,20,10,1,0.5,0,128,255
```


[0]: 'image.jpg' (640x427x1x3)

## polygon

Built-in command

### Arguments:

- `N>=1,x1[%],y1[%],...,xN[%],yN[%],_opacity,_pattern,_color1,...`

## Description:

Draw specified colored N-vertices polygon on selected images.

`pattern` is an hexadecimal number starting with `0x` which can be omitted even if a color is specified. If a pattern is specified, the polygon is drawn outlined instead of filled.

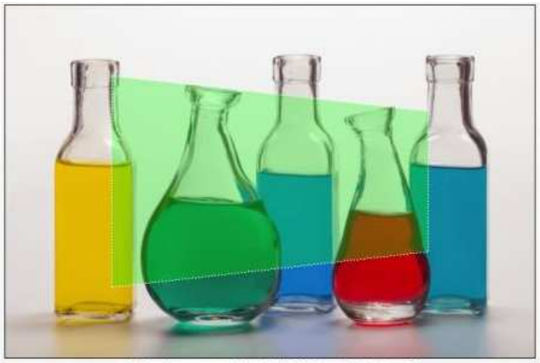
## Default values:

`opacity=1`, `pattern=(undefined)` and `color1=0`.

## Examples of use:

### • Example #1

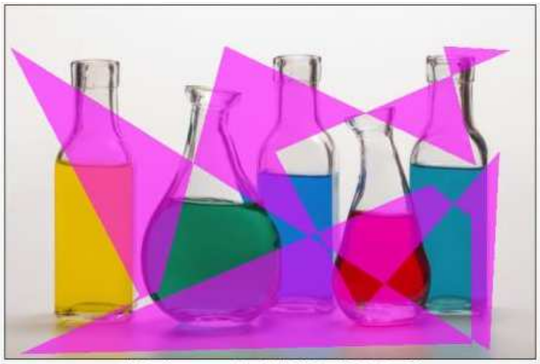
```
image.jpg polygon 4,20%,20%,80%,30%,80%,70%,20%,80%,0.3,0,255,0
polygon 4,20%,20%,80%,30%,80%,70%,20%,80%,1,0xCCCCCCCC,255
```



[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg 2,16,1,1,'u(x?{h}:{w})' polygon[-2] {h},{^},0.6,255,0,255
remove[-1]
```



[0]: 'image.jpg' (640x427x1x3)

# polygonize

## Arguments:

- `_warp_amplitude>=0,_smoothness[%]>=0,_min_area[%]>=0,_resolution_x[%]>0,_resc`

## Description:

Apply polygon effect on selected images.

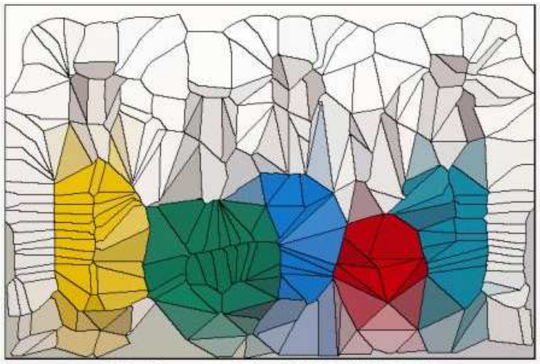
## Default values:

`warp_amplitude=300`, `smoothness=2%`, `min_area=0.1%`, `resolution_x=resolution_y=10%`.

## Example of use:

```
image.jpg +polygonize 100,10 fill[-1] "I!=J(1) || I!=J(0,1)?
[0,0,0]:I"
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# portrait

## Arguments:

- `_size>0`

## Description:

Input random portrait image of specified size, retrieved from Internet.

## Default values:

`size=800`.

# pose3d

## Arguments:

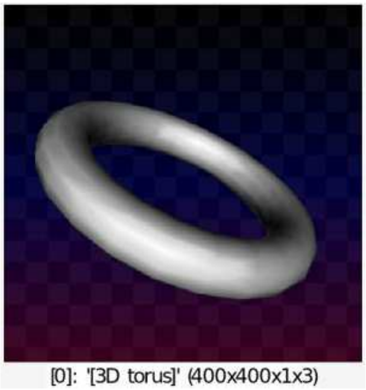
- `p1,...,p12`

## Description:

Apply 3D pose matrix to selected 3D objects.

## Example of use:

```
torus3d 100,20 pose3d 0.152437,1.20666,-0.546366,0,
-0.535962,0.559129,1.08531,0,1.21132,0.0955431,0.548966,0,0,0,-206,1
snapshot3d 400
```



[0]: '[3D torus]' (400x400x1x3)

# poster_edges

## Arguments:

- `0<=_edge_threshold<=100,0<=_edge_shade<=100,_edge_thickness>=0,_edge_antialia`

## Description:

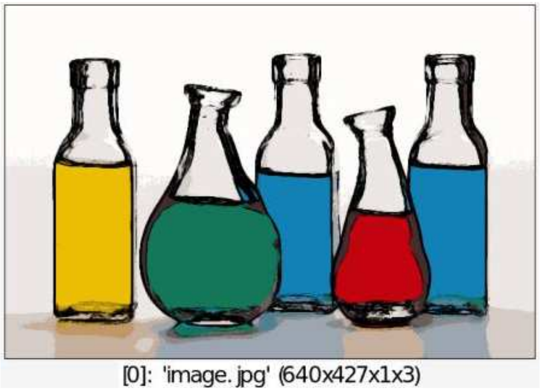
Apply poster edges effect on selected images.

## Default values:

`edge_threshold=40`, `edge_shade=5`, `edge_thickness=0.5`, `edge_antialiasing=10`, `posterization_level=12` and `posterization_antialiasing=0`.

## Example of use:

```
image.jpg poster_edges ,
```



[0]: 'image.jpg' (640x427x1x3)

## poster_hope

### Arguments:

- `_smoothness>=0`

### Description:

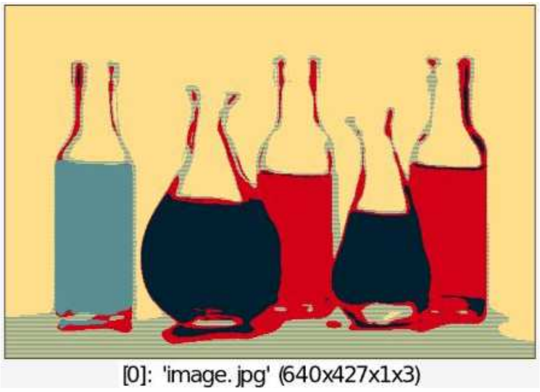
Apply Hope stencil poster effect on selected images.

### Default values:

`smoothness=3`.

### Example of use:

```
image.jpg poster_hope ,
```



[0]: 'image.jpg' (640x427x1x3)

## pow

Built-in command

### Arguments:

- `value[%]` or
- `[image]` or

- `'formula'` or
- `(no arg)`

## Description:

Raise selected images to the power of specified value, image or mathematical expression, or compute the pointwise sequential powers of selected images.

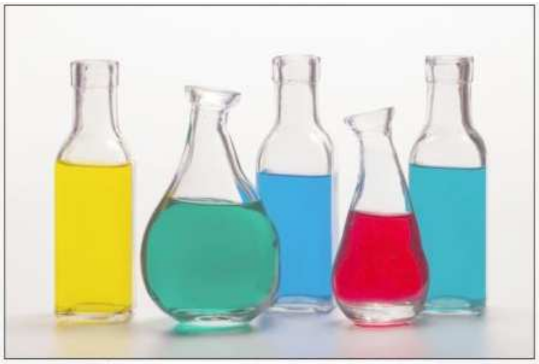
(*equivalent to shortcut command* `^`).

## Examples of use:

### • Example #1

```
image.jpg div 255 +pow 0.5 mul 255
```

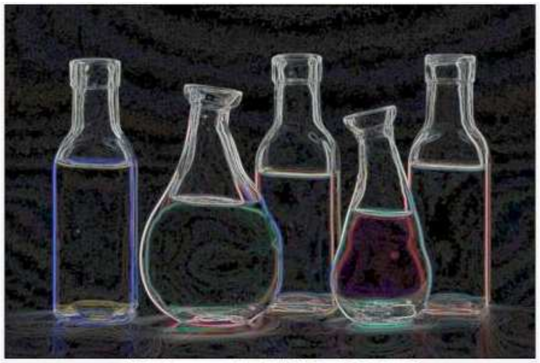
[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
image.jpg gradient pow 2 add pow 0.2
```



[0]: 'image.jpg' (640x427x1x3)

# poweriteration

## Arguments:

- `_nb_eigenvectors>0,_epsilon>0,_max_iter>0`

## Description:

Compute the `nb_eigenvectors` largest eigenvectors of the selected symmetric matrices,

using the power iteration algorithm.

## Default values:

`nb_eigenvectors=1`, `epsilon=1e-5` and `max_iter=100`.

---

# premula

### No arguments

## Description:

Convert selected images with normal colors to premultiplied alpha colors.

After conversion, alpha channel of resulting images has value in [0,1] range.

## See also:

**ipremula**.

---

# primitives3d

## Arguments:

- `mode`

## Description:

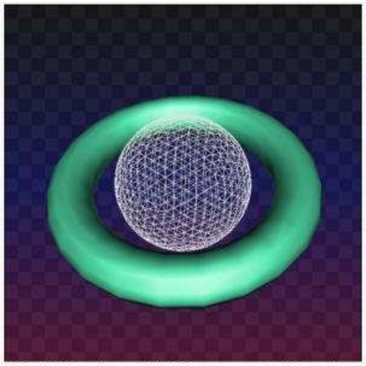
Convert primitives of selected 3D objects.

(*equivalent to shortcut command* `p3d`).

`mode` can be ***`{ 0:points | 1:outlines | 2:non-textured }`***.

## Example of use:

```
sphere3d 30 primitives3d 1 torus3d 50,10 color3d[-1] ${-rgb} add3d
```



[0]: '[3D sphere]' (930 vert., 2208 prim.)

---

# print

### No arguments

## Description:

Print information on selected images, on the standard error ( `stderr` ).

(*equivalent to shortcut command* `p`).

When invoked with a `+` prefix (i.e. `+print` ), the command outputs on `stdout` rather than on `stderr`.

---

# progress

**Built-in command**

## Arguments:

- `0<=value<=100` or
- `-1`

## Description:

Set the progress index of the current processing pipeline.

This command is useful only when G'MIC is used by an embedding application.

---

# projections3d

## Arguments:

- `_x[%],_y[%],_z[%],_is_bounding_box={ 0 | 1 },nb_subdivisions>0`

## Description:

Generate 3D xy,xz,yz projection planes from specified volumetric images.

## Default values:

`x=y=z=50%`, `is_bounding_box=1` and `nb_subdividions=5`

---

# pseudogray

## Arguments:

- `_max_increment>=0,_JND_threshold>=0,_bits_depth>0`

## Description:

Generate pseudogray colormap with specified increment and perceptual threshold.

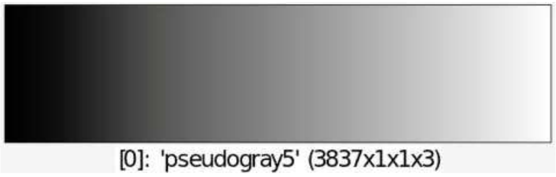
If `JND_threshold` is 0, no perceptual constraints are applied.

## Default values:

`max_increment=5`, `JND_threshold=2.3` and `bits_depth=8`.

## Example of use:

```
pseudogray 5
```



---

# psnr

## Arguments:

- `[reference],_max_value>0`

## Description:

Return PSNR (Peak Signal-to-Noise Ratio) between selected images and specified reference image.

This command does not modify the images. It returns a value or a list of values in the status.

## Default values:

`max_value=255`.

# psnr_matrix

## Arguments:

- `_max_value>0`

## Description:

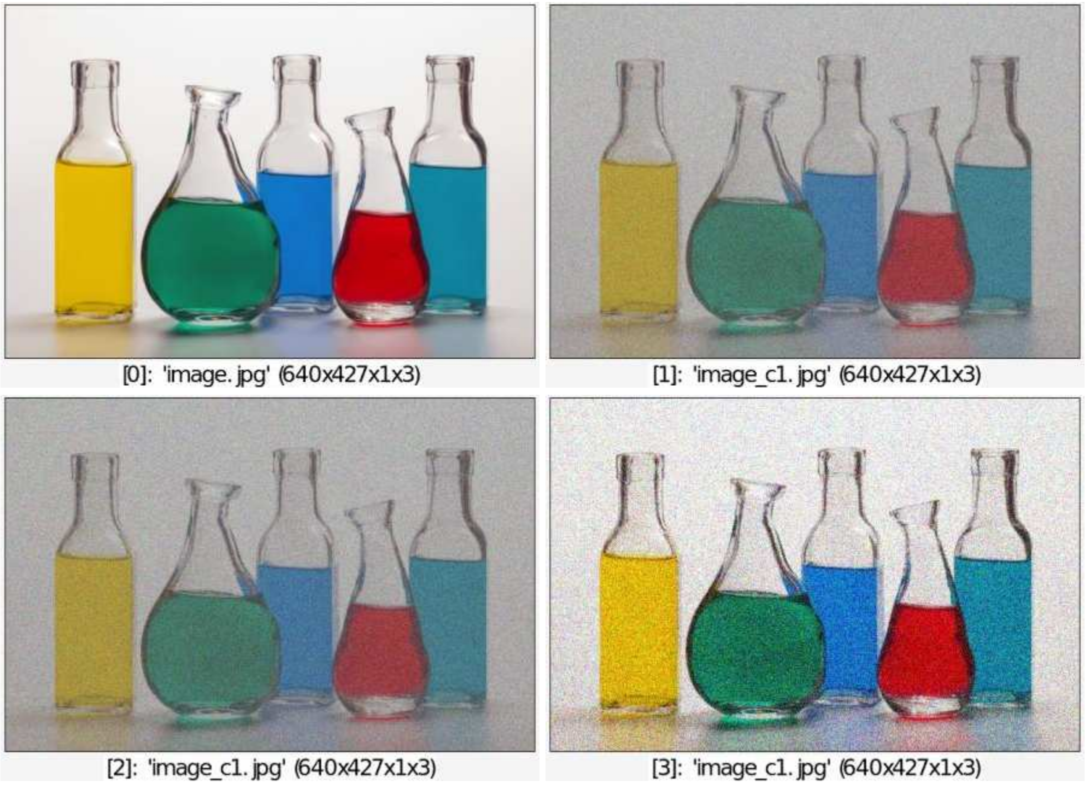
Compute PSNR (Peak Signal-to-Noise Ratio) matrix between selected images.

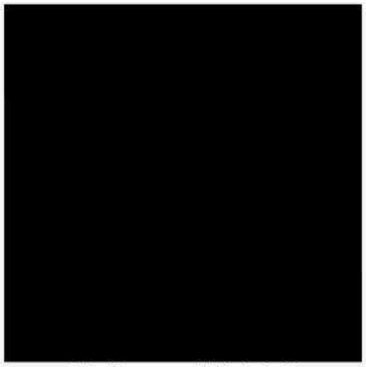
## Default values:

`max_value=255`.

## Example of use:

```
image.jpg +noise 30 +noise[0] 35 +noise[0] 38 cut. 0,255 +psnr_matrix
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)
[2]: 'image_c1.jpg' (640x427x1x3)
[3]: 'image_c1.jpg' (640x427x1x3)



[4]: '[unnamed]' (4x4x1x1)

# puzzle

## Arguments:

- `_width>0,_height>0,_M>=1,_N>=1,_curvature,_centering,_connectors_variability,`

## Description:

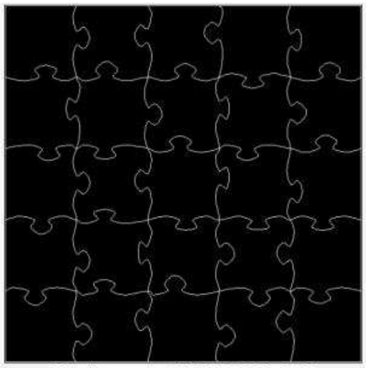
Input puzzle binary mask with specified size and geometry.

## Default values:

`width=height=512`, `M=N=5`, `curvature=0.5`, `centering=0.5`, `connectors_variability=0.5` and `resolution=64`.

## Example of use:

```
puzzle ,
```



[0]: '[unnamed]' (512x512x1x1)

# pyramid3d

## Arguments:

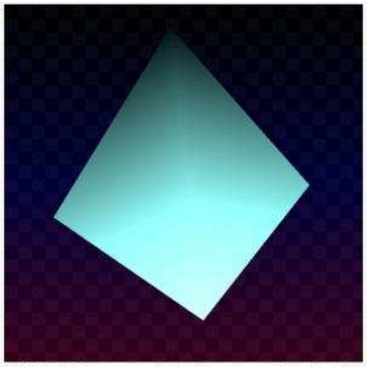
- `width,height`

## Description:

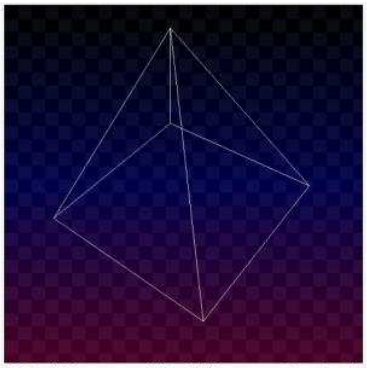
Input 3D pyramid at (0,0,0), with specified geometry.

## Example of use:

```
pyramid3d 100,-100 +primitives3d 1 color3d[-2] ${-rgb}
```



[0]: '[3D pyramid]' (5 vert., 5 prim.)



[1]: '[3D pyramid]_c1' (5 vert., 8 prim.)

---

# quadrangle3d

## Arguments:

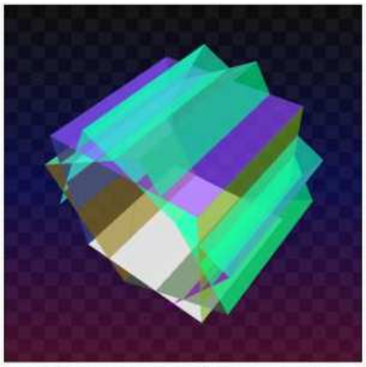
- `x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3`

## Description:

Input 3D quadrangle at specified coordinates.

## Example of use:

```
quadrangle3d -10,-10,10,10,-10,10,10,10,10,-10,10,10 repeat 10 {
+rotate3d[-1] 0,1,0,30 color3d[-1] ${-rgb},0.6 } add3d mode3d 2
```



[0]: '[3D quadrangle]' (44 vert., 11 prim.)

---

# quadratize_tiles

## Arguments:

- `M>0,_N>0`

## Description:

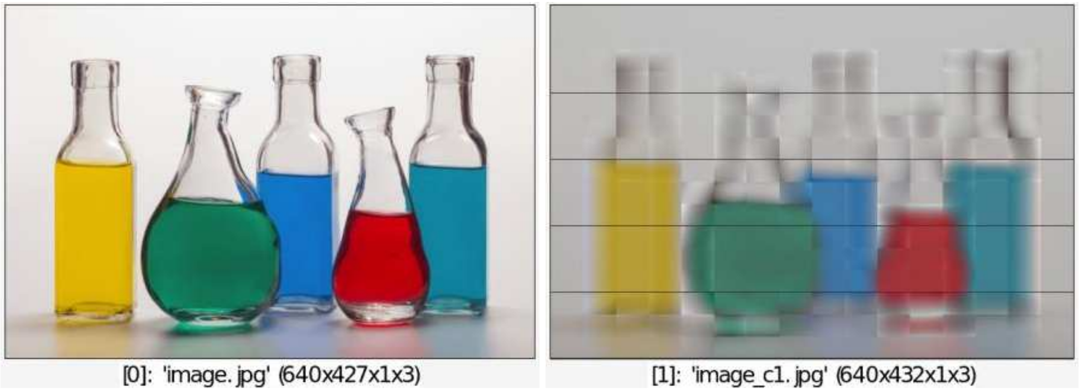
Quadratize MxN tiles on selected images.

## Default values:

`N=M`.

## Example of use:

```
image.jpg +quadratize_tiles 16
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x432x1x3)

---

# quantize

## Arguments:

- `nb_levels>=1,_keep_values={ 0 | 1 },_quantization_type={ -1:median-cut | 0:k-means | 1:uniform }`

## Description:

Quantize selected images.

## Default values:

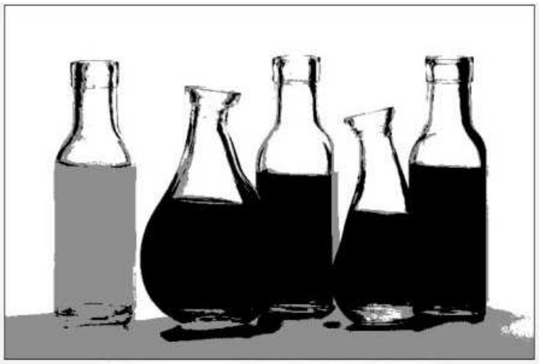
`keep_values=1` and `quantization_type=0`.

## Examples of use:

- **Example #1**

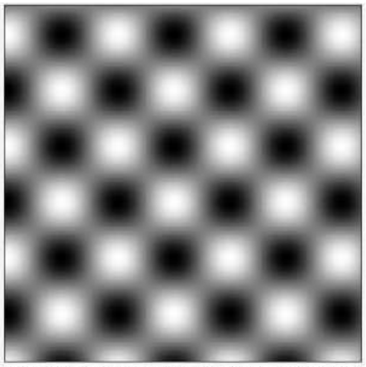
```
image.jpg luminance +quantize 3
```

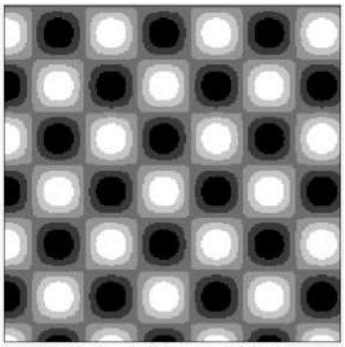
[0]: 'image.jpg' (640x427x1x1)

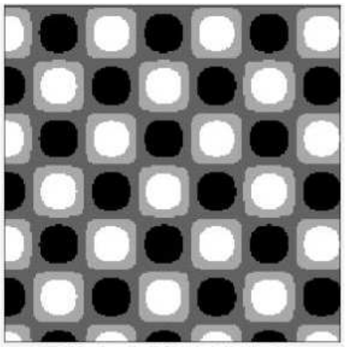

[1]: 'image_c1.jpg' (640x427x1x1)

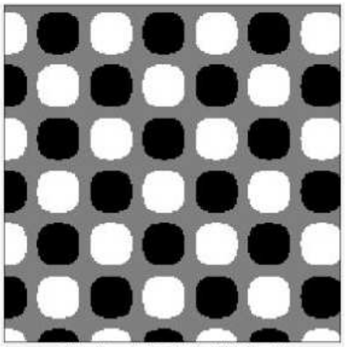
- **Example #2**

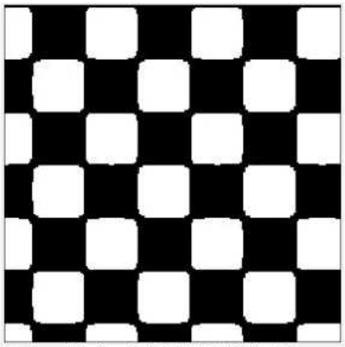
```
200,200,1,1,'cos(x/10)*sin(y/10)' +quantize[0] 6 +quantize[0] 4
+quantize[0] 3 +quantize[0] 2
```


[0]: '[cos(x/10)*sin(y/10)]' (200x200x1x1)


[1]: '[cos(x/10)*sin(y/10)]_c1'
(200x200x1x1)


[2]: '[cos(x/10)*sin(y/10)]_c1'
(200x200x1x1)


[3]: '[cos(x/10)*sin(y/10)]_c1'
(200x200x1x1)


[4]: '[cos(x/10)*sin(y/10)]_c1'
(200x200x1x1)

# quantize_area

## Arguments:

- `_min_area>0`

## Description:

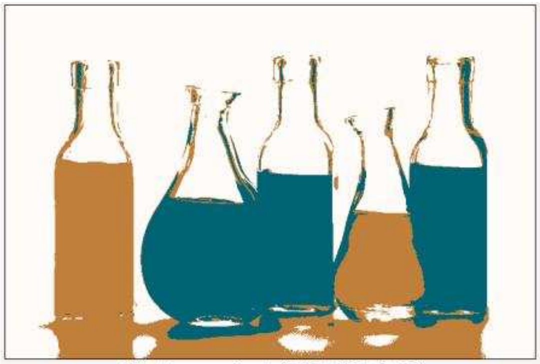
Quantize selected images such that each flat region has an area greater or equal to `min_area`.

### Default values:

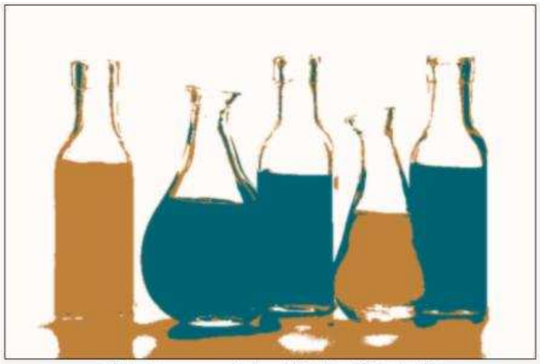
`min_area=10`.

### Example of use:

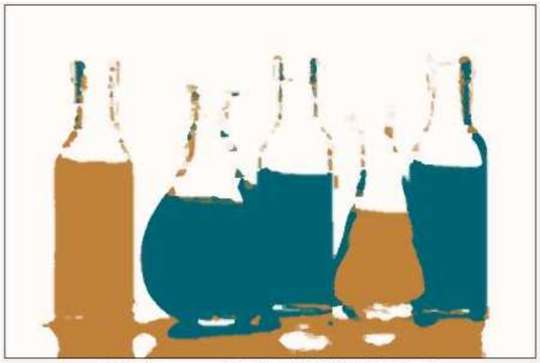
```
image.jpg quantize 3 +blur 1 round[-1] +quantize_area[-1] 2
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c13.jpg' (640x427x1x3)

## quit

**Built-in command**

### No arguments

### Description:

Quit G'MIC interpreter.

(*equivalent to shortcut command* `q`).

## quiver

### Arguments:

- `[function_image],_sampling[%]>0,_factor>=0,_is_arrow={ 0 | 1 },_opacity,_color1,...`

## Description:

Draw specified 2D vector/orientation field on selected images.

## Default values:

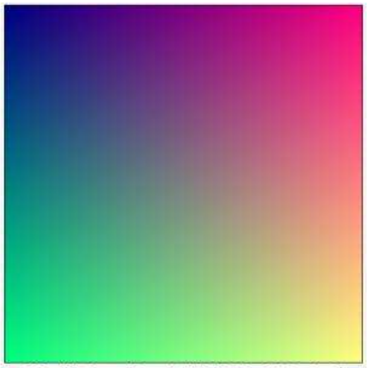
`sampling=5%`, `factor=1`, `is_arrow=1`, `opacity=1`, `pattern=(undefined)`

and `color1=0`.

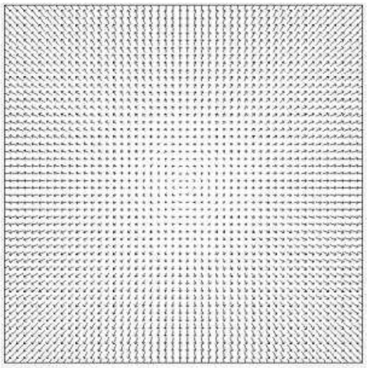
## Examples of use:

### • Example #1

```
100,100,1,2,'!c?x-w/2:y-h/2' 500,500,1,3,255 quiver[-1] [-2],10
```



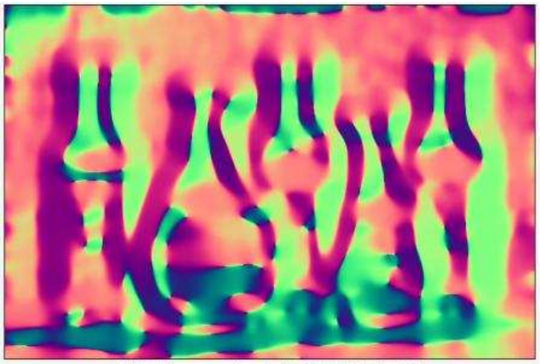
[0]: '[!c?x-w/2:y-h/2]' (100x100x1x2)



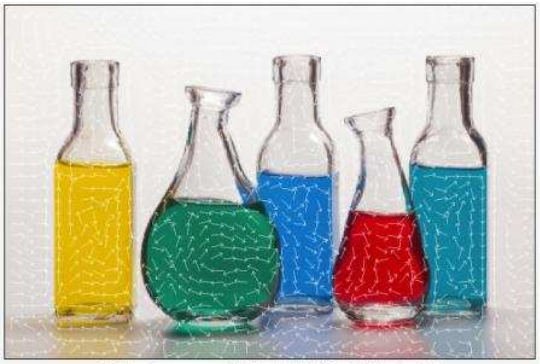
[1]: '[255]' (500x500x1x3)

### • Example #2

```
image.jpg +rescale2d ,600 luminance[0] gradient[0] mul[1] -1
reverse[0,1] append[0,1] c blur[0] 8 orientation[0] quiver[1] [0],
20,1,1,0.8,255
```



[0]: 'image.jpg' (640x427x1x2)



[1]: 'image_c1.jpg' (899x600x1x3)

# rad2deg

**No arguments**

## Description:

Convert pointwise angle values of selected images, from radians to degrees (apply `i*180/pi`).

---

# raindrops

## Arguments:

- `_amplitude,_density>=0,_wavelength>=0,_merging_steps>=0`

## Description:

Apply raindrops deformation on selected images.

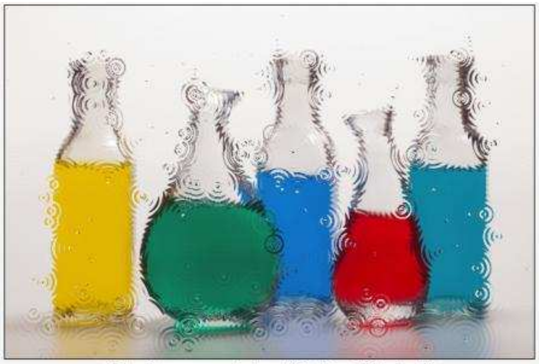
## Default values:

`amplitude=80`, `density=0.1`, `wavelength=1` and `merging_steps=0`.

## Example of use:

```
image.jpg +raindrops ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# rand

Built-in command

## Arguments:

- `{ value0[%] | [image0] },_{ value1[%] | [image1] },_[pdf],_precision` or
- `[image]`

## Description:

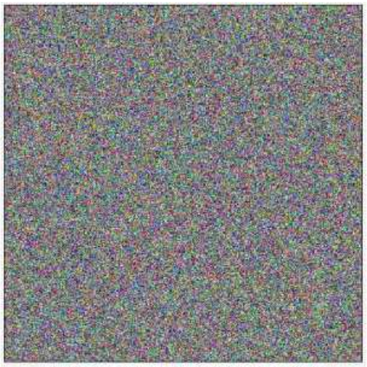
Fill selected images with random values in the specified range.

If no `[pdf]` (probability density function) is specified, random values follow a uniform distribution. Argument `precision` tells about the number of distinct values that can be generated when a `[pdf]` is specified.

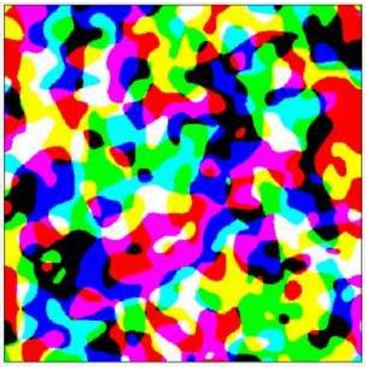
## Examples of use:

### • Example #1

```
400,400,1,3 rand -10,10 +blur 10 sign[-1]
```



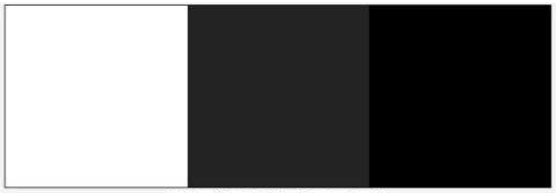
[0]: '[unnamed]' (400x400x1x3)



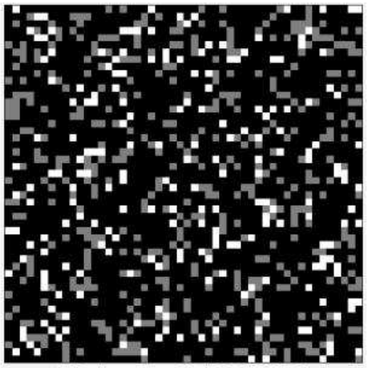
[1]: '[unnamed]_c1' (400x400x1x3)

### • Example #2

```
(8,2,1) 50,50 rand[-1] 0,255,[-2]
```



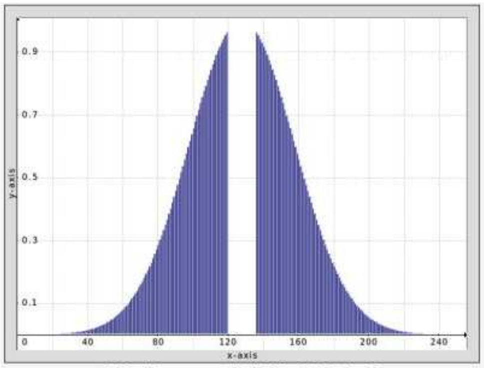
[0]: '(8,2,1)' (3x1x1x1)



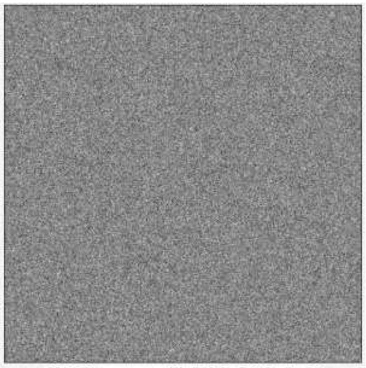
[1]: '[unnamed]' (50x50x1x1)

### • Example #3

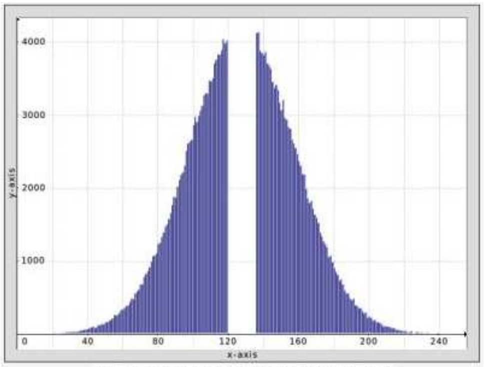
```
256 gaussian[-1] 30 line[-1] 47%,0,53%,0,1,0 500,500 rand[-1] 0,255,
[-2] +histogram[-1] 256 display_graph[0,2] 640,480,3,0
```



[0]: '[unnamed]' (640x480x1x3)



[1]: '[unnamed]' (500x500x1x1)



[2]: '[unnamed]_c1' (640x480x1x3)

# rand_sum

## Arguments:

- `sum>0,_random_function`

## Description:

Fill selected images with strictly positive, random, integer values, that sums to `sum`.

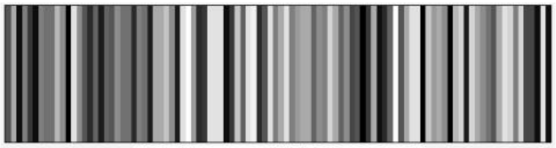
For each image, `sum` must be greater or equal than `width*height*depth*spectrum`.

## Default values:

`random_function=u`.

## Example of use:

```
100 rand_sum 1000
```



[0]: '[unnamed]' (100x1x1x1)

# random3d

## Arguments:

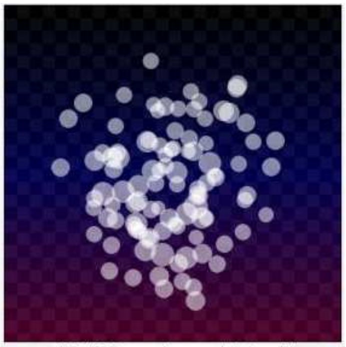
- `nb_points>=0`

## Description:

Input random 3D point cloud in [0,1]^3.

## Example of use:

```
random3d 100 circles3d 0.1 opacity3d 0.5
```



[0]: '[3D random pointcloud]'
(200 vert., 100 prim.)

# random_clut

## Arguments:

- `_seed = { >=0 | -1 }`

## Description:

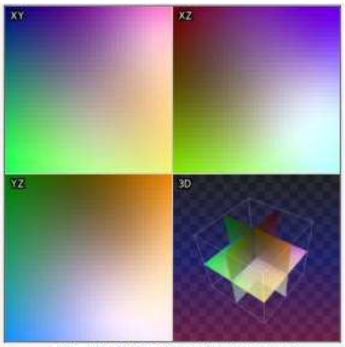
Generate a 33x33x33 random 3D color LUT.

If specified `seed` is positive, it is used as a seed for the random number generator @cli : (so that using the same seed will return the same CLUT).

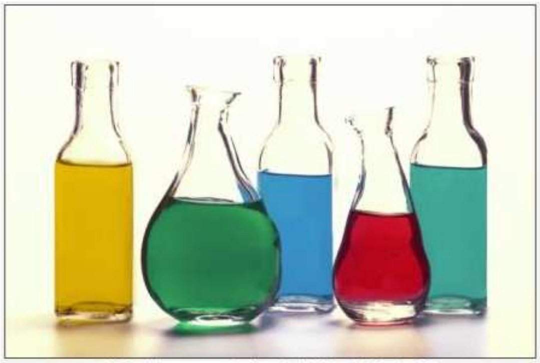
## Example of use:

```
image.jpg random_clut +map_clut.. .
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '(256;0^256;255^256;1)' (33x33x33x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# random_clut

## Arguments:

- `_seed = { >=0 | -1 }`

## Description:

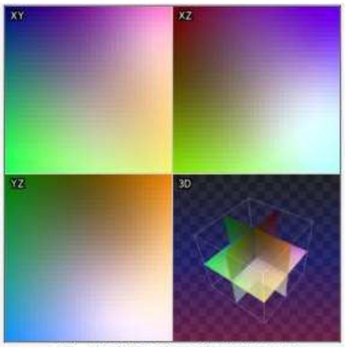
Generate a 33x33x33 random 3D color LUT.

If specified `seed` is positive, it is used as a seed for the random number generator @cli : (so that using the same seed will return the same CLUT).

## Example of use:

```
image.jpg random_clut +map_clut.. .
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '(256;0^256;255^256;1)' (33x33x33x3)

[2]: 'image_c1.jpg' (640x427x1x3)

---

# random_pattern

## Arguments:

- `_width>0,_height>0,_min_detail_level>=0`

## Description:

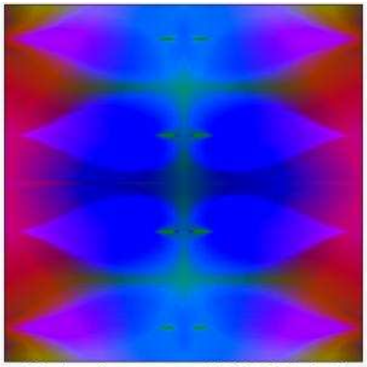
Insert a new RGB image of specified size at the end of the image list, rendered with a random pattern.

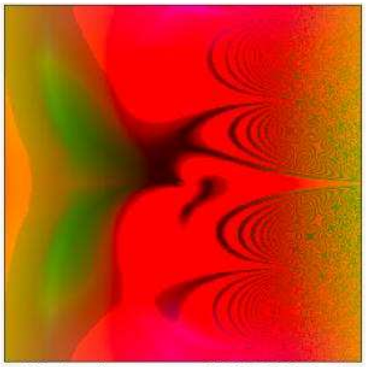
## Default values:

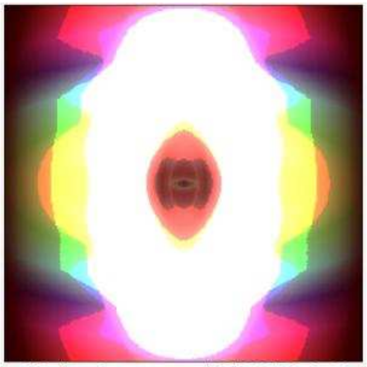
`width=height=512` and `min_detail_level=2`.

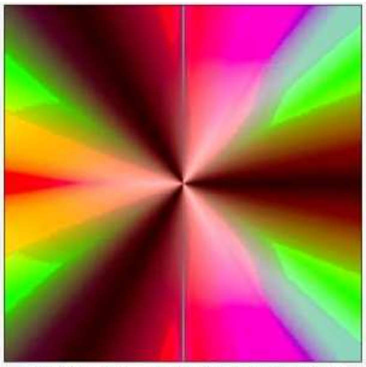
## Example of use:

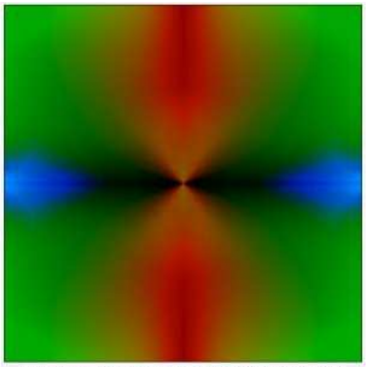
```
repeat 6 { random_pattern 256 }
```

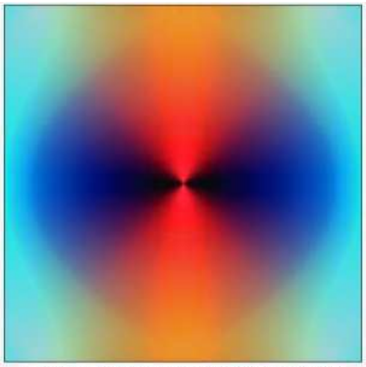

[0]: 'random_pattern' (256x256x1x3)


[1]: 'random_pattern' (256x256x1x3)


[2]: 'random_pattern' (256x256x1x3)


[3]: 'random_pattern' (256x256x1x3)


[4]: 'random_pattern' (256x256x1x3)


[5]: 'random_pattern' (256x256x1x3)

# rbf

## Arguments:

- `dx,_x0,_x1,_phi(r)` or
- `dx,dy,_x0,_y0,_x1,_y1,_phi(r)` or
- `dx,dy,dz,x0,y0,z0,x1,y1,z1,phi(r)`

## Description:

Reconstruct 1D/2D or 3D image from selected sets of keypoints, by RBF-interpolation.

A set of keypoints is represented by a vector-valued image, where each pixel represents a single keypoint.
Vector components of a keypoint have the following meaning:

- For 1D reconstruction: [ x_k, f1(k),...fN(k) ].
- For 2D reconstruction: [ x_k,y_k, f1(k),...,fN(k) ].
- For 3D reconstruction: [ x_k,y_k,z_k, f1(k),...,fN(k) ].

Values `x_k`, `y_k` and `z_k` are the spatial coordinates of keypoint `k`.
Values `f1(k),..,fN(k)` are the `N` components of the vector value of keypoint `k`.
The command reconstructs an image with specified size `dx'x'dy'x'dz`, with `N` channels.

## Default values:

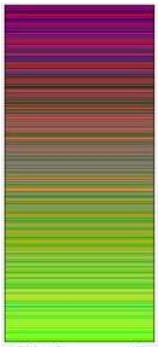
`x0=y0=z0=0`, `x1=dx-1`, `y1=dy-1`, `z1=dz-1`, `phi(r)=r^2*log(1e-5+r)`.

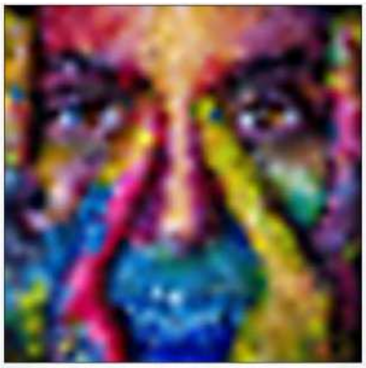
## Examples of use:

- **Example #1**

```
sample colorful,400 100%,100% noise_poissondisk. 10 1,{is},1,5
eval[-2] "begin(p=0);i?(I[#-1,p++]=[x,y,I(#0)])" to_rgb[1] mul[0,1]
dilate_circ[0] 5 +rbf[-1] {0,[w,h]} c[-1] 0,255
```

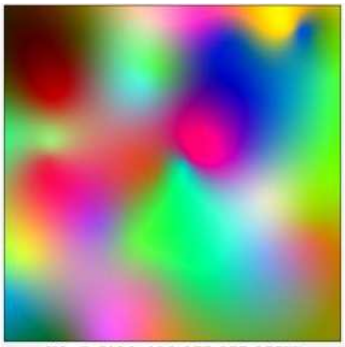
[0]: 'colorful' (400x400x1x3)


[1]: '[unnamed]' (1x1013x1x5)


[2]: '[unnamed]_c1' (400x400x1x3)

- **Example #2**

```
32,1,1,5,u([400,400,255,255,255]) rbf 400,400 c 0,255
```


[0]: '[u([400,400,255,255,255])]' (400x400x1x3)

# rectangle

## Arguments:

- `x0[%],y0[%],x1[%],y1[%],_opacity,_pattern,_color1,...`

## Description:

Draw specified colored rectangle on selected images.

`pattern` is an hexadecimal number starting with `0x` which can be omitted even if a color is specified. If a pattern is specified, the rectangle is drawn outlined instead of filled.

## Default values:

`opacity=1`, `pattern=(undefined)` and `color1=0`.

## Example of use:

```
image.jpg repeat 30 { rectangle {u(100)}%,{u(100)}%,{u(100)}%,
{u(100)}%,0.3,${-rgb} }
```

[0]: 'image.jpg' (640x427x1x3)

# red_eye

## Arguments:

- `0<=_threshold<=100,_smoothness>=0,0<=attenuation<=1`

## Description:

Attenuate red-eye effect in selected images.

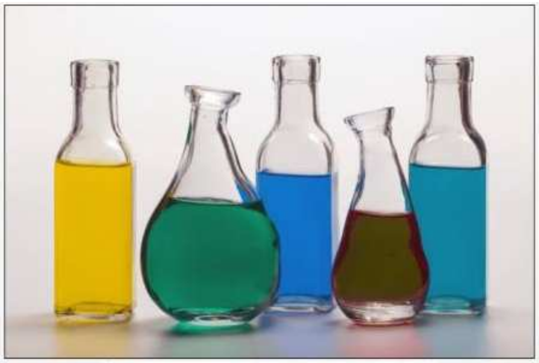
## Default values:

`threshold=75`, `smoothness=3.5` and `attenuation=0.1`.

## Example of use:

```
image.jpg +red_eye ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# register_nonrigid

## Arguments:

- `[destination],_smoothness>=0,_precision>0,_nb_scale>=0`

## Description:

Register selected source images with specified destination image, using non-rigid warp.

## Default values:

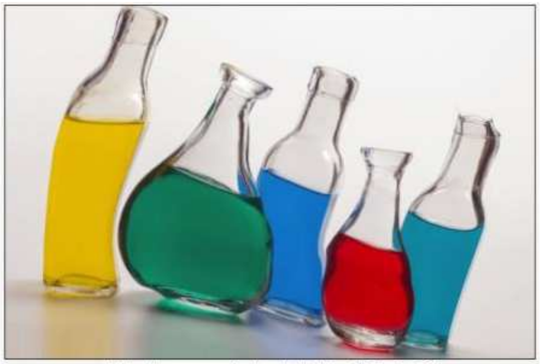
`smoothness=0.2`, `precision=6` and `nb_scale=0(auto)`.

## Example of use:

```
image.jpg +rotate 20,1,1,50%,50% +register_nonrigid[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# register_rigid

## Arguments:

- `[destination],_smoothness>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Register selected source images with specified destination image, using rigid warp (shift).

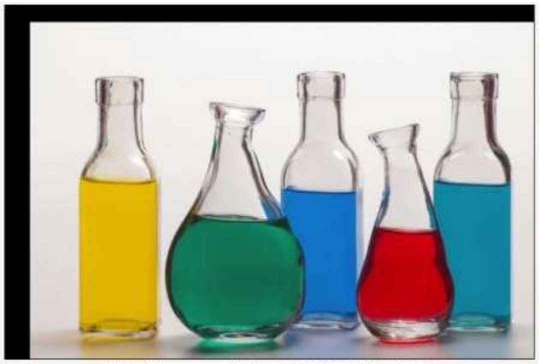
## Default values:

`smoothness=0.1%` and `boundary_conditions=0`.

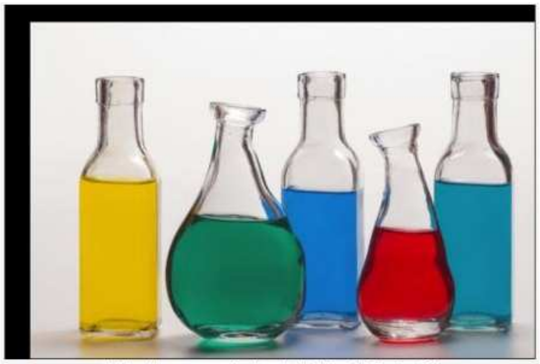
## Example of use:

```
image.jpg +shift 30,20 +register_rigid[0] [1]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# remove

Built-in command

### No arguments

## Description:

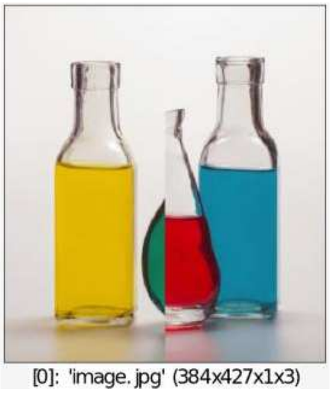
Remove selected images.

(*equivalent to shortcut command* `rm`).

## Examples of use:

### • Example #1

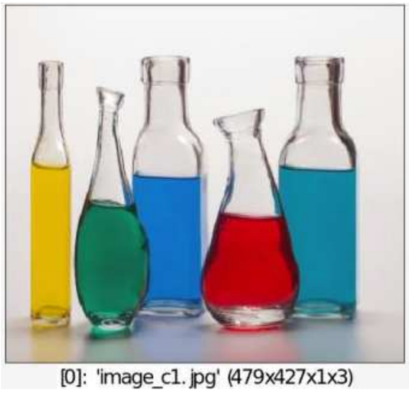
```
image.jpg split x remove[30%-70%] append x
```



[0]: 'image.jpg' (384x427x1x3)

- **Example #2**

```
image.jpg split x remove[0-50%:2] append x
```



[0]: 'image_c1.jpg' (479x427x1x3)

---

# remove_copymark

**No arguments**

## Description:

Remove copymark suffix in names of selected images.

---

# remove_duplicates

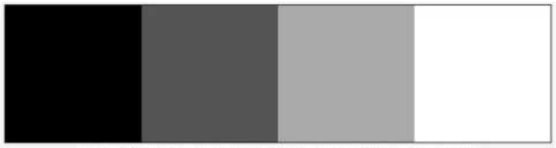
**No arguments**

## Description:

Remove duplicates images in the selected images list.

## Example of use:

```
(1,2,3,4,2,4,3,1,3,4,2,1) split x remove_duplicates append x
```



[0]: '(1,2,3,4,2,4,3,1,3,4,2,1)' (4x1x1x1)

# remove_empty

**No arguments**

## Description:

Remove empty images in the selected image list.

# remove_hotpixels

## Arguments:

- `_mask_size>0, _threshold[%]>0`

## Description:

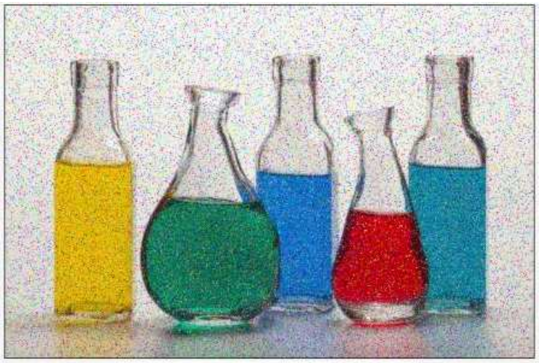
Remove hot pixels in selected images.

## Default values:

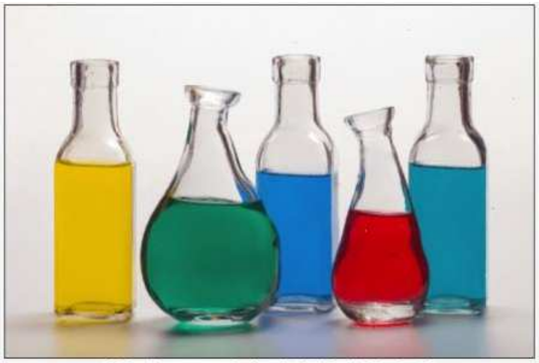
`mask_size=3` and `threshold=10%`.

## Example of use:

```
image.jpg noise 10,2 +remove_hotpixels ,
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# remove_named

## Arguments:

- `"name1","name2",...`

## Description:

Remove all images with specified names from the list of images.

Does nothing if no images with those names exist.

(*equivalent to shortcut command* `rmn`).

---

# remove_opacity

**No arguments**

## Description:

Remove opacity channel of selected images.

---

# remove_pixels

## Arguments:

- `number_of_pixels[%]>=0`

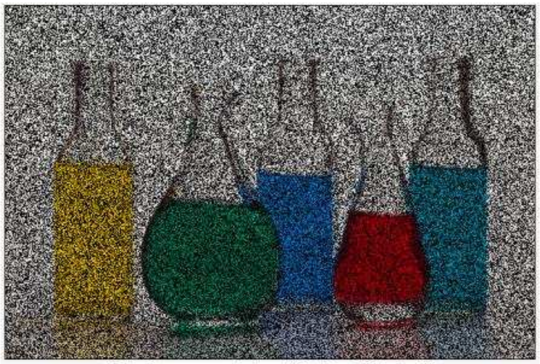
## Description:

Remove specified number of pixels (i.e. set them to 0) from the set of non-zero pixels in selected images.

## Example of use:

```
image.jpg +remove_pixels 50%
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

# repeat

Built-in command

## Arguments:

- `nb_iterations`

## Description:

Start `nb_iterations` iterations of a `repeat...done` block.

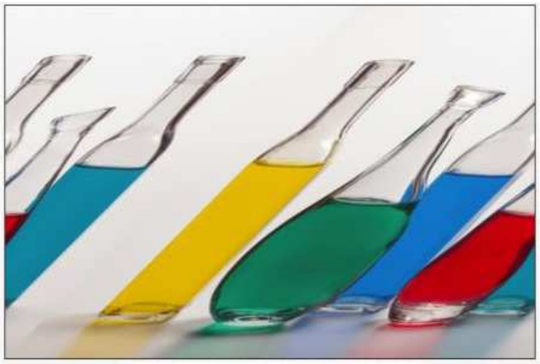
`nb_iterations` is a mathematical expression that will be evaluated.

This command has a **tutorial page**.

## Examples of use:

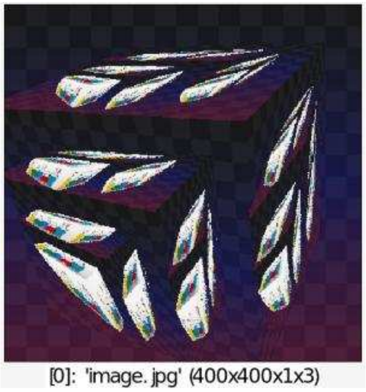
### • Example #1

```
image.jpg split y repeat $! n=$> shift[$n] $<,0,0,0,2 done append y
```


[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg mode3d 2 repeat 4 imagecube3d rotate3d 1,1,0,40 snapshot3d
400,1.4 done
```



# replace

## Arguments:

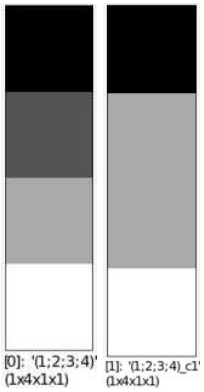
- `source,target`

## Description:

Replace pixel values in selected images.

## Example of use:

```
(1;2;3;4) +replace 2,3
```



# replace_color

## Arguments:

- `tolerance[%]>=0,smoothness[%]>=0,src1,src2,...,dest1,dest2,...`

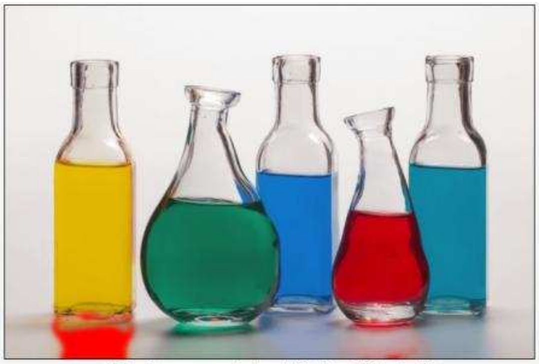
## Description:

Replace pixels from/to specified colors in selected images.

### Example of use:

```
image.jpg +replace_color 40,3,204,153,110,255,0,0
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## replace_inf

### Arguments:

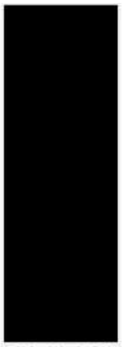
- `_expression`

### Description:

Replace all infinite values in selected images by specified expression.

### Example of use:

```
(0;1;2) log +replace_inf 2
```



[0]: '(0;1;2)' (1x3x1x1)

[1]: '(0;1;2)_c1' (1x3x1x1)

## replace_infnan

### Arguments:

- `_expression`

## Description:

Replace all NaN and infinite values in selected images by specified expression.

# replace_nan

## Arguments:

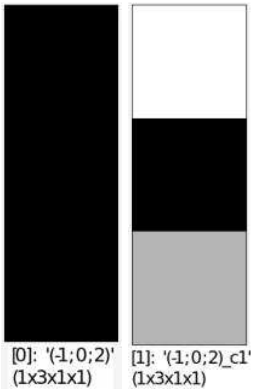
- `_expression`

## Description:

Replace all NaN values in selected images by specified expression.

## Example of use:

```
(-1;0;2) sqrt +replace_nan 2
```



# replace_seq

## Arguments:

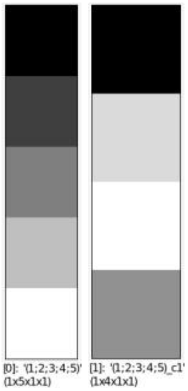
- `"search_seq","replace_seq"`

## Description:

Search and replace a sequence of values in selected images.

## Example of use:

```
(1;2;3;4;5) +replace_seq "2,3,4","7,8"
```



# replace_str

## Arguments:

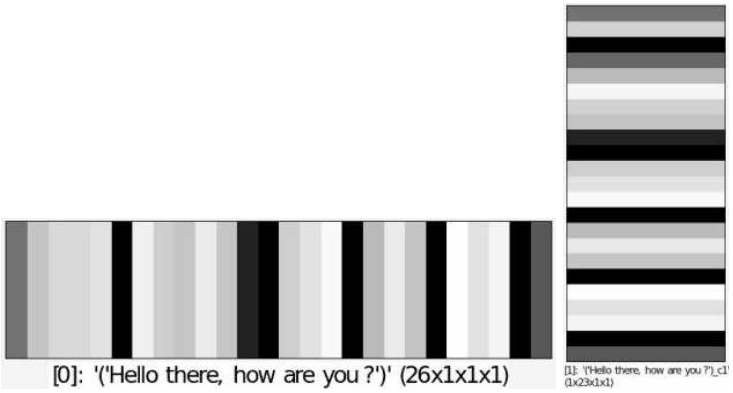
- `"search_str","replace_str"`

## Description:

Search and replace a string in selected images (viewed as strings, i.e. sequences of character codes).

## Example of use:

```
('"Hello there, how are you ?"') +replace_str "Hello there","Hi David"
```



# rescale2d

## Arguments:

- `_width[%]={ 0:any | >0 },_height[%]={ 0:any | >0 },-1=<_interpolation<=6,_mode={ 0:inside | 1:padded-inside | 2:outside | 3:cropped-outside }`

## Description:

Resize selected 2D images while preserving aspect ratio.

`interpolation` can be ***{ -1:status only | 0:none | 1:nearest | 2:average | 3:linear | 4=grid | 5=bicubic | 6=lanczos }***.
When `interpolation==-1`, image size is actually not modified, but the size that would have been used for the last selected image is returned in the status value.
Each resized image size is computed according to the specified `mode`:

- If `mode==0`, image size is **at most** `(width,height)`.
- If `mode==1` or `mode==3`, image size is **exactly** `(width,height)`.
- If `mode==2`, image size is **at least** `(width,height)`.

(*equivalent to shortcut command* `rs`).

## Default values:

`width=height=0`, `interpolation=2` and `mode=0`.

---

# rescale3d

## Arguments:

- `_width[%]={ 0:any | >0 },_height[%]={ 0:any | >0 },_depth[%]={ 0:any | >0 },-1=<_interpolation<=6,_mode={ 0:inside | 1:padded-inside | 2:outside | 3` or
- `cropped-outside }`

## Description:

Resize selected 3D images while preserving aspect ratio.

`interpolation` can be ***{ -1:status only | 0:none | 1:nearest | 2:average | 3:linear | 4=grid | 5=bicubic | 6=lanczos }***.
When `interpolation==-1`, image size is actually not modified, but the size that would have been used for the last selected image is returned in the status value.
Each resized image size is computed according to the specified `mode`:

- If `mode==0`, image size is **at most** `(width,height)`.
- If `mode==1` or `mode==3`, image size is **exactly** `(width,height)`.
- If `mode==2`, image size is **at least** `(width,height)`.

(*equivalent to shortcut command* `rs3d`).

## Default values:

`width=height=depth=0`, `interpolation=2` and `mode=0`.

---

# reset

**No arguments**

## Description:

Reset global parameters of the interpreter environment.

---

# resize

**Built-in command**

## Arguments:

- `{[image_w] | width>0[%]},_{[image_h] | height>0[%]},_{[image_d] | depth>0[%]},_{[image_s] | spectrum>0[%]},_interpolation,_boundary_conditions,_ax,_ay,_az,_ac`

## Description:

Resize selected images with specified geometry.

(*equivalent to shortcut command* `r`).

`interpolation` can be ***{ -1:none (memory content) | 0:none | 1:nearest | 2:average | 3:linear | 4=grid | 5=bicubic | 6=lanczos }***.
`boundary_conditions` has different meanings, according to the chosen `interpolation` mode :
. When 'interpolation=={ -1 | 1 | 2 | 4 }', `boundary_conditions` is meaningless.
. When `interpolation==0`, `boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
. When 'interpolation=={ 3 | 5 | 6 }', `boundary_conditions` can be ***{ 0:none | 1:neumann }***.
`ax,ay,az,ac` set the centering along each axis when 'interpolation=0 or 4'
(set to `0` by default, must be defined in range [0,1]).

## Default values:

`interpolation=1`, `boundary_conditions=0` and `ax=ay=az=ac=0`.

## Example of use:

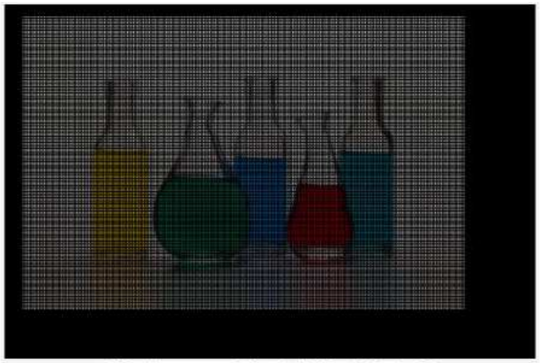
```
image.jpg +resize[-1] 256,128,1,3,2 +resize[-1] 120%,120%,
1,3,0,1,0.5,0.5 +resize[-1] 120%,120%,1,3,0,0,0.2,0.2 +resize[-1]
[0],[0],1,3,4
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (256x128x1x3)

[2]: 'image_c2.jpg' (307x154x1x3)

[3]: 'image_c3.jpg' (368x185x1x3)



[4]: 'image_c4.jpg' (640x427x1x3)

# resize_as_image

## Arguments:

- `[reference],_interpolation,_boundary_conditions,_ax,_ay,_az,_ac`

## Description:

Resize selected images to the geometry of specified [reference] image.

(*equivalent to shortcut command* `ri`).

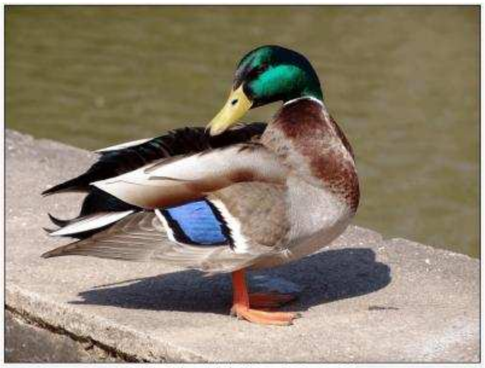
## Default values:

`interpolation=1`, `boundary_conditions=0` and `ax=ay=az=ac=0`.

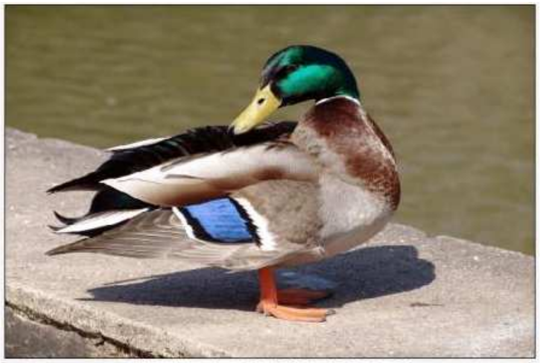
## Example of use:

```
image.jpg sample duck +resize_as_image[-1] [-2]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'duck' (640x480x1x3)



[2]: 'duck_c1' (640x427x1x3)

# resize_mn

## Arguments:

- `width[%]>=0,_height[%]>=0,_depth[%]>=0,_B_value,_C_value`

## Description:

Resize selected images with Mitchell-Netravali filter (cubic).

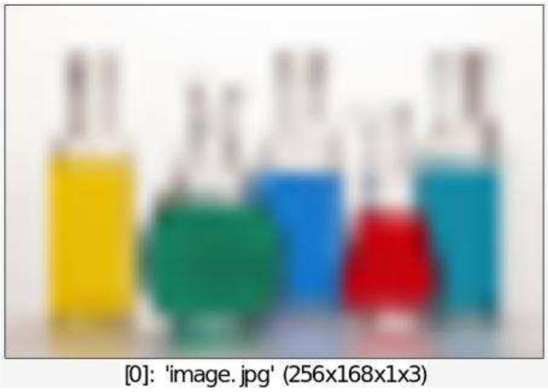
For details about the method, see: **https://de.wikipedia.org/wiki/Mitchell-Netravali-Filter**.

## Default values:

`height=100%`, `depth=100%`, `B=0.3333` and `C=0.3333`.

## Example of use:

```
image.jpg rescale2d 32 resize_mn 800%,800%
```


[0]: 'image.jpg' (256x168x1x3)

---

# resize_pow2

## Arguments:

- `_interpolation,_boundary_conditions,_ax,_ay,_az,_ac`

## Description:

Resize selected images so that each dimension is a power of 2.

`interpolation` can be ***{ -1:none (memory content) | 0:none | 1:nearest | 2:average | 3:linear | 4:grid | 5:bicubic | 6:lanczos }***.
`boundary_conditions` has different meanings, according to the chosen `interpolation` mode :
. When 'interpolation=={ -1 | 1 | 2 | 4 }', `boundary_conditions` is meaningless.
. When `interpolation==0`, `boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
. When 'interpolation=={ 3 | 5 | 6 }', `boundary_conditions` can be ***{ 0:none | 1:neumann }***.
`ax,ay,az,ac` set the centering along each axis when `interpolation=0`
(set to `0` by default, must be defined in range [0,1]).

## Default values:

`interpolation=0`, `boundary_conditions=0` and `ax=ay=az=ac=0`.

## Example of use:

```
image.jpg +resize_pow2[-1] 0
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (1024x512x1x3)

## retinex

### Arguments:

- `_value_offset>0,_colorspace={ hsi | hsv | lab | lrgb | rgb | ycbcr },0<=_min_cut<=100,0<=_max_cut<=100,_sigma_low>0,_sigma_mid>0,_sigma_high>0`

### Description:

Apply multi-scale retinex algorithm on selected images to improve color consistency.

(as described in the page **http://www.ipol.im/pub/art/2014/107/**).

### Default values:

`offset=1`, `colorspace=hsv`, `min_cut=1`, `max_cut=1`, `sigma_low=15`, `sigma_mid=80` and `sigma_high=250`.

## return

Built-in command

**No arguments**

### Description:

Return from current custom command.

## reverse

Built-in command

**No arguments**

### Description:

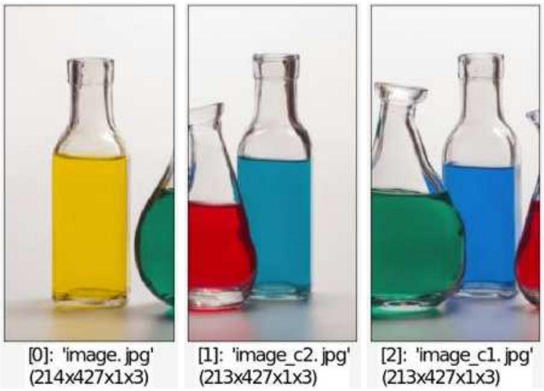
Reverse positions of selected images.

(*equivalent to shortcut command* `rv`).

## Examples of use:

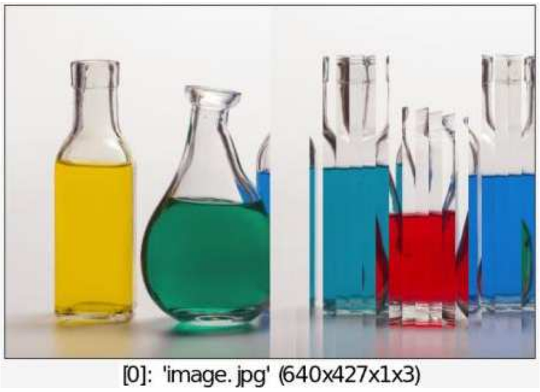
### • Example #1

```
image.jpg split x,3 reverse[-2,-1]
```



### • Example #2

```
image.jpg split x,-16 reverse[50%-100%] append x
```



---

# reverse3d

### No arguments

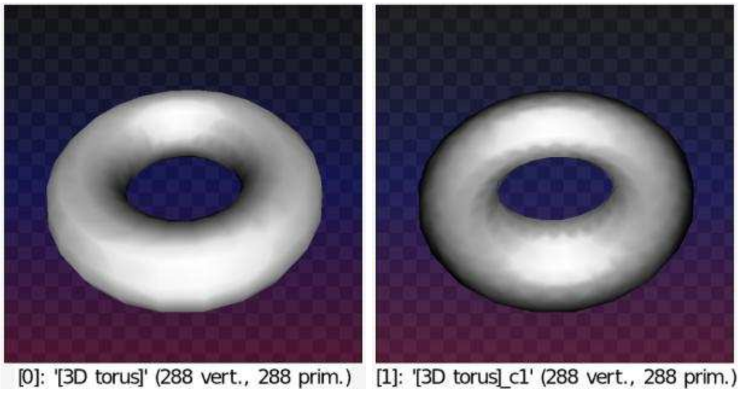
## Description:

Reverse primitive orientations of selected 3D objects.

(*equivalent to shortcut command* `rv3d`).

## Example of use:

```
torus3d 100,40 double3d 0 +reverse3d
```


[0]: '[3D torus]' (288 vert., 288 prim.)
[1]: '[3D torus]_c1' (288 vert., 288 prim.)

---

# rgb

**No arguments**

## Description:

Return a random int-valued RGB color.

---

# rgb2bayer

## Arguments:

- `_start_pattern=0,_color_grid=0`

## Description:

Transform selected color images to RGB-Bayer sampled images.

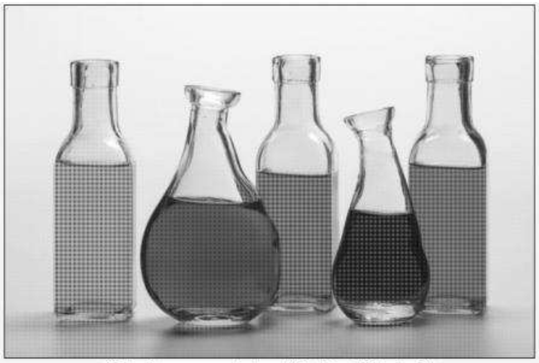
## Default values:

`start_pattern=0` and `color_grid=0`.

## Example of use:

```
image.jpg +rgb2bayer 0
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x1)

## rgb2cmy

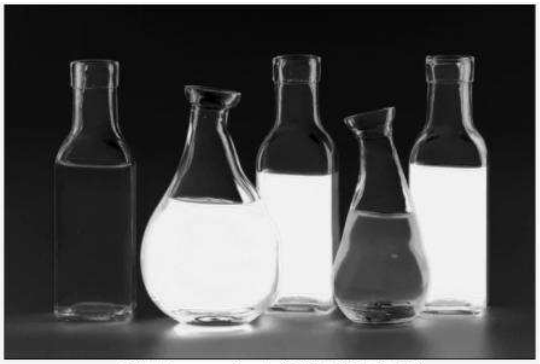
**No arguments**

### Description:

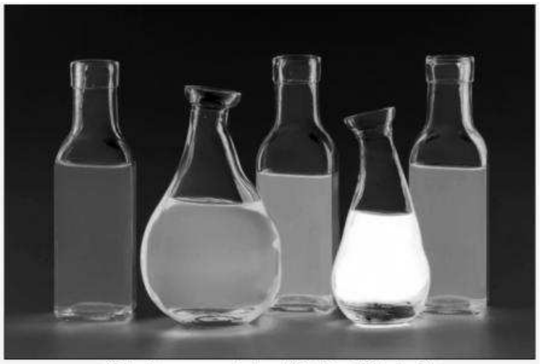
Convert color representation of selected images from RGB to CMY.

### Example of use:

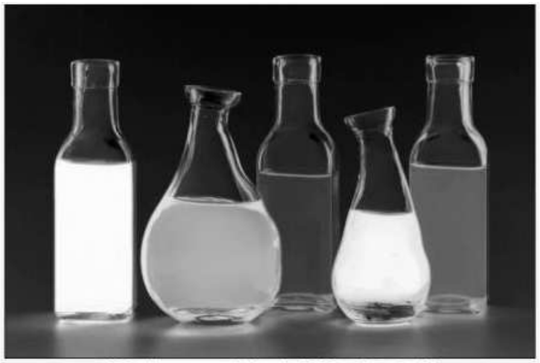
```
image.jpg rgb2cmy split c
```



[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

## rgb2cmyk

**No arguments**

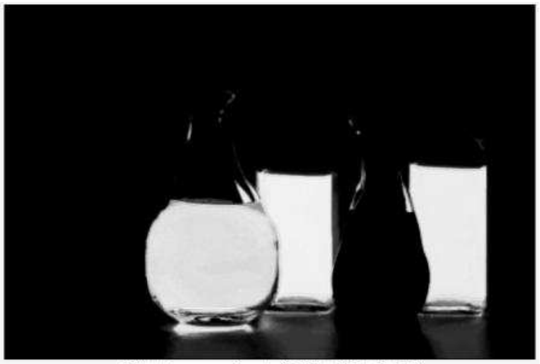
## Description:

Convert color representation of selected images from RGB to CMYK.

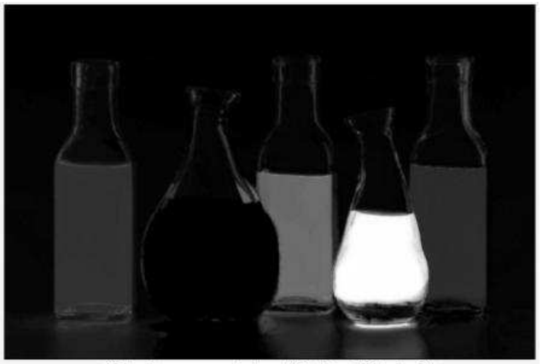
## Examples of use:

### • Example #1

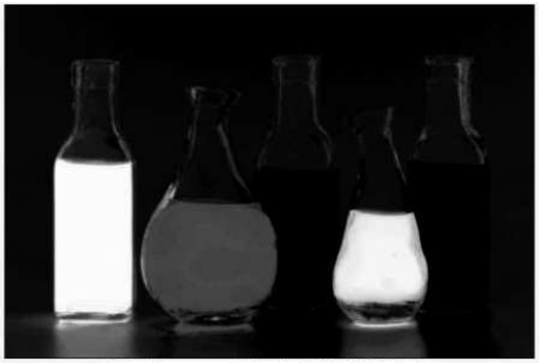
```
image.jpg rgb2cmyk split c
```



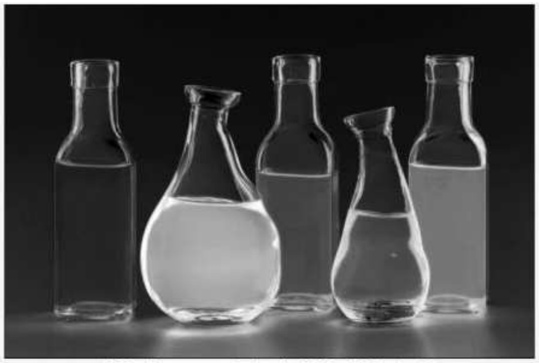
[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



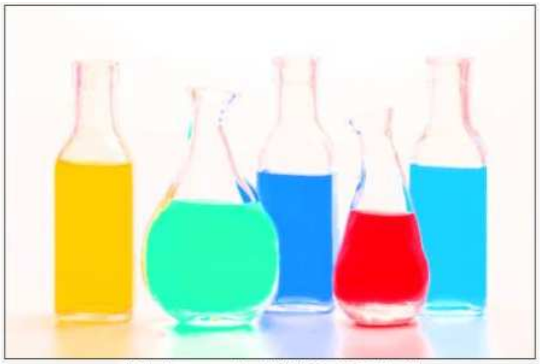
[2]: 'image_c2.jpg' (640x427x1x1)



[3]: 'image_c3.jpg' (640x427x1x1)

### • Example #2

```
image.jpg rgb2cmyk split c fill[3] 0 append c cmyk2rgb
```



[0]: 'image.jpg' (640x427x1x3)

---

# rgb2hcy

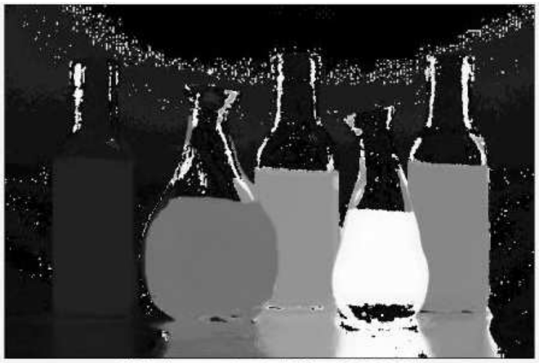
**No arguments**

## Description:

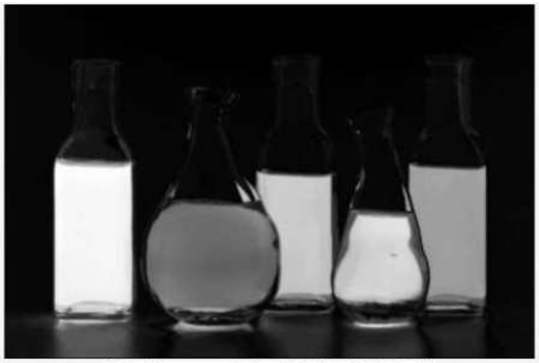
Convert color representation of selected images from RGB to HCY.

## Example of use:

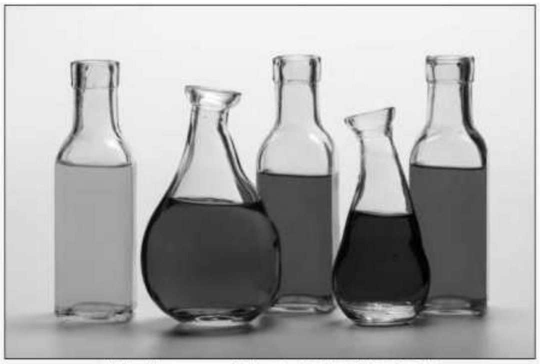
```
image.jpg rgb2hcy split c
```



[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

---

# rgb2hsi

**No arguments**

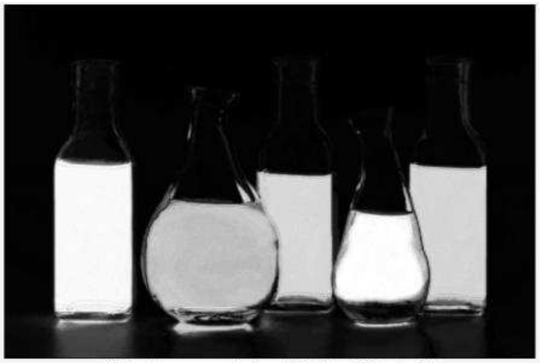
## Description:

Convert color representation of selected images from RGB to HSI.

## Example of use:

```
image.jpg rgb2hsi split c
```

[0]: 'image.jpg' (640x427x1x1)


[1]: 'image_c1.jpg' (640x427x1x1)

[2]: 'image_c2.jpg' (640x427x1x1)

## rgb2hsi8

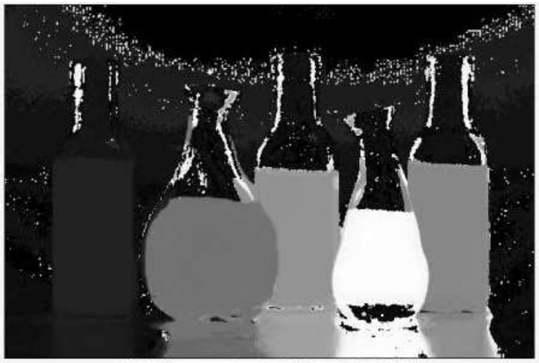
**No arguments**

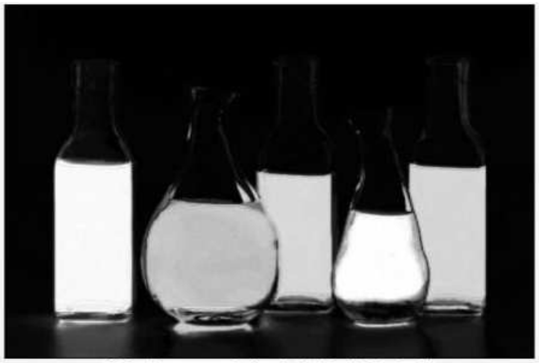
### Description:

Convert color representation of selected images from RGB to HSI8.

### Example of use:

```
image.jpg rgb2hsi8 split c
```


[0]: 'image.jpg' (640x427x1x1)


[1]: 'image_c1.jpg' (640x427x1x1)

[2]: 'image_c2.jpg' (640x427x1x1)

---

# rgb2hsl

**No arguments**

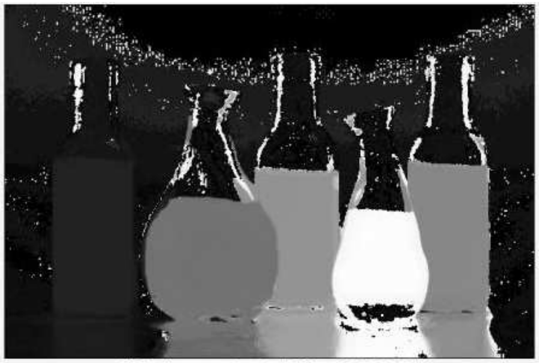
## Description:

Convert color representation of selected images from RGB to HSL.

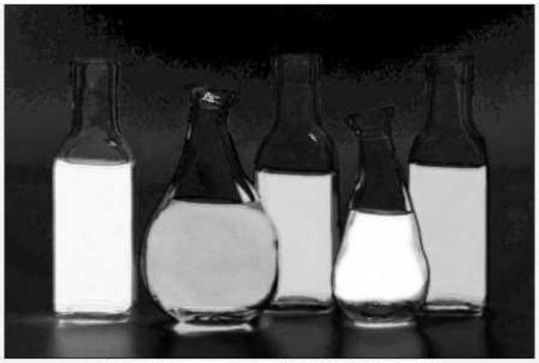
## Examples of use:

- **Example #1**

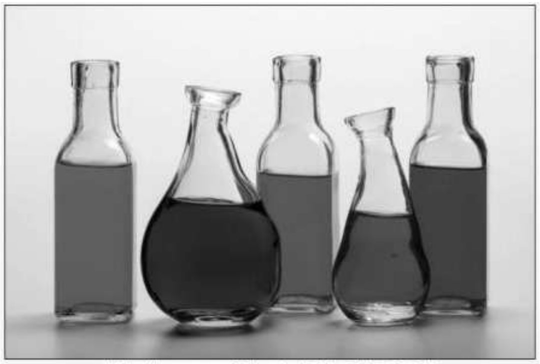
```
image.jpg rgb2hsl split c
```



[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)

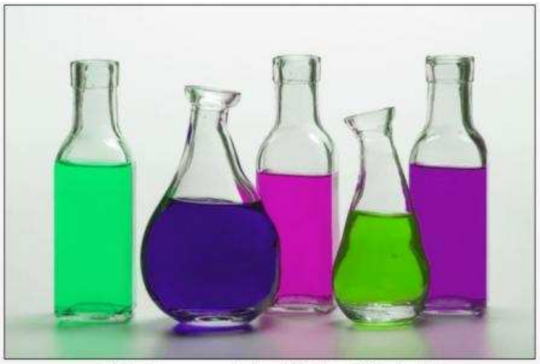


[2]: 'image_c2.jpg' (640x427x1x1)

- **Example #2**

```
image.jpg rgb2hsl +split c add[-3] 100 mod[-3] 360 append[-3--1] c
hsl2rgb
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

## rgb2hsl8

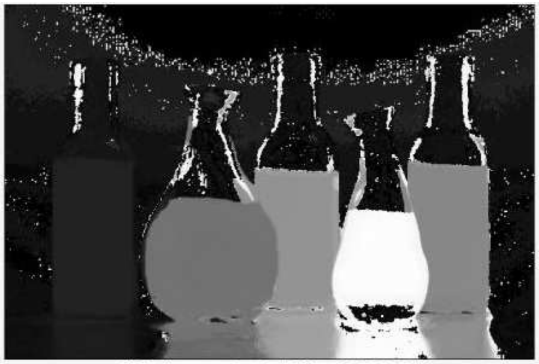
**No arguments**

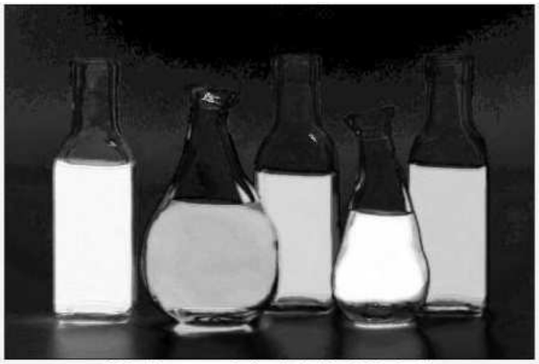
### Description:

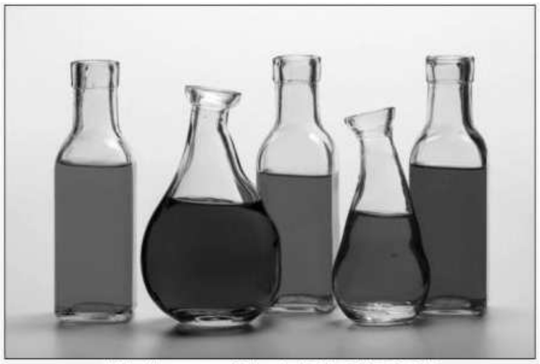
Convert color representation of selected images from RGB to HSL8.

### Example of use:

```
image.jpg rgb2hsl8 split c
```


[0]: 'image.jpg' (640x427x1x1)


[1]: 'image_c1.jpg' (640x427x1x1)


[2]: 'image_c2.jpg' (640x427x1x1)

# rgb2hsv

**No arguments**

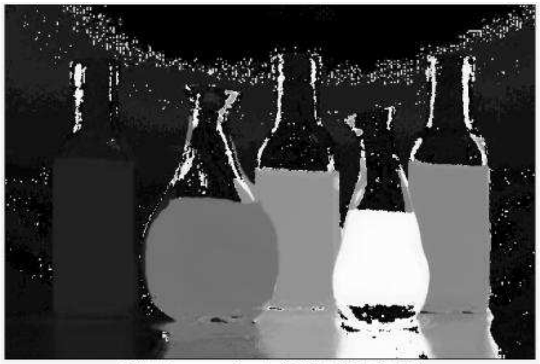
## Description:

Convert color representation of selected images from RGB to HSV.

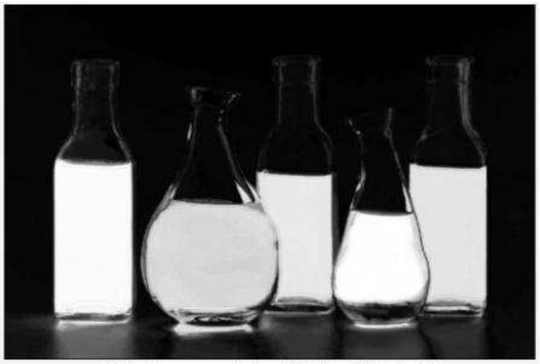
## Examples of use:

### • Example #1

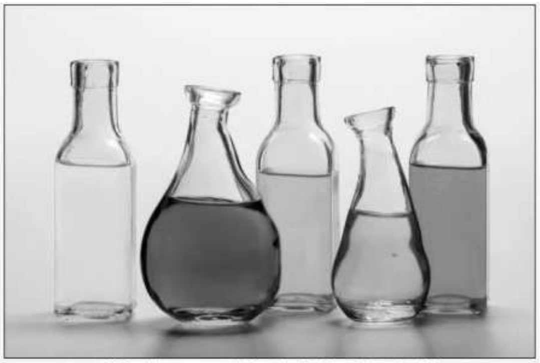
```
image.jpg rgb2hsv split c
```



[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

### • Example #2

```
image.jpg rgb2hsv +split c add[-2] 0.3 cut[-2] 0,1 append[-3--1] c
hsv2rgb
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

## rgb2hsv8

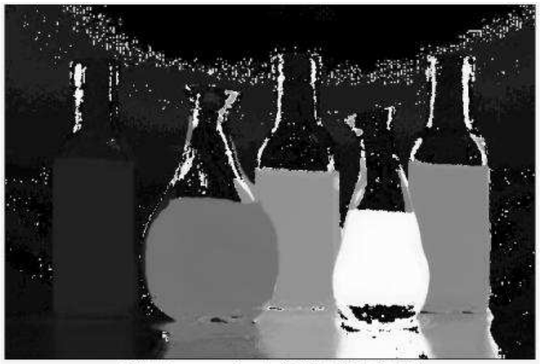
**No arguments**

### Description:

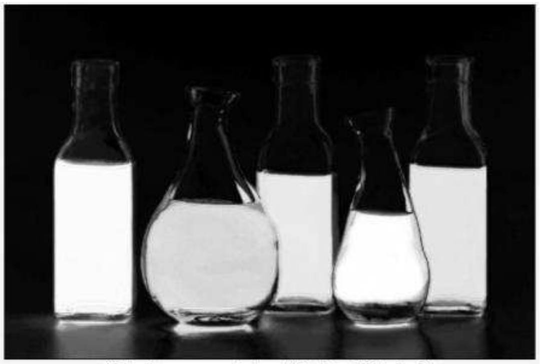
Convert color representation of selected images from RGB to HSV8.

### Example of use:

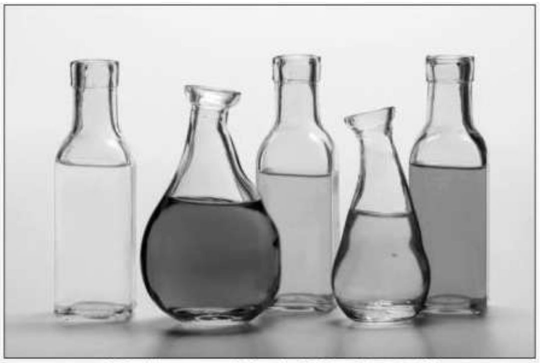
```
image.jpg rgb2hsv8 split c
```



[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

## rgb2int

**No arguments**

## Description:

Convert color representation of selected images from RGB to INT24 scalars.

## Example of use:

```
image.jpg rgb2int
```

[0]: 'image.jpg' (640x427x1x1)

---

# rgb2jzazbz

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from RGB to Jzazbz.

## Default values:

`illuminant=2`.

---

# rgb2lab

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from RGB to Lab.

### Default values:

`illuminant=2`.

## rgb2lab8

### Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

### Description:

Convert color representation of selected images from RGB to Lab8.

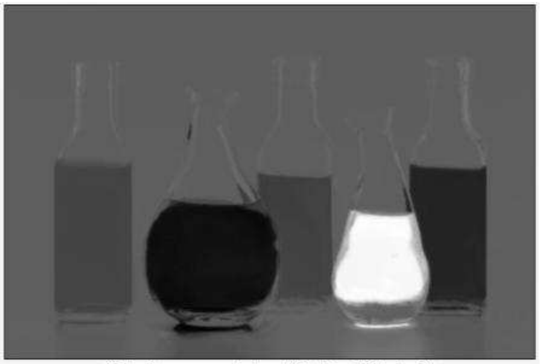
### Default values:

`illuminant=2`.

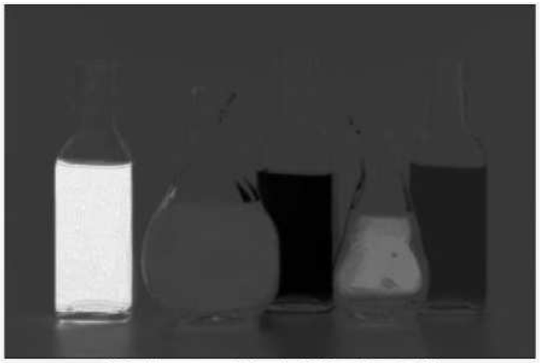
### Example of use:

```
image.jpg rgb2lab8 split c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

## rgb2lch

### Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

### Description:

Convert color representation of selected images from RGB to Lch.

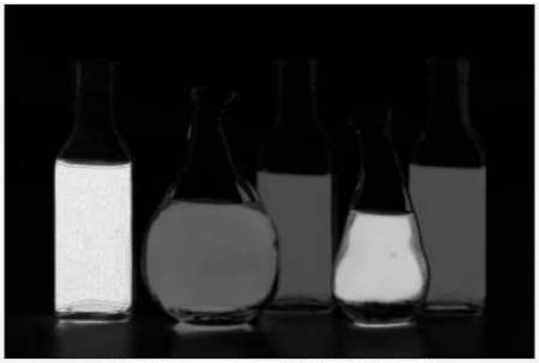
### Default values:

`illuminant=2`.

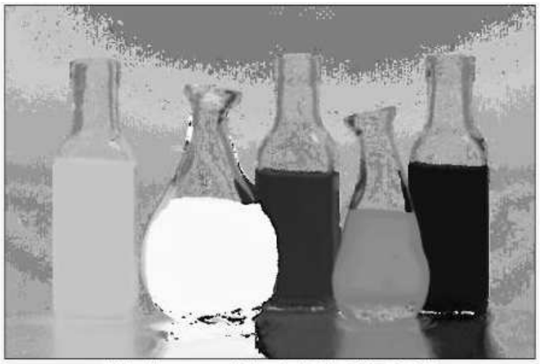
### Example of use:

```
image.jpg rgb2lch split c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

## rgb2lch8

### Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

### Description:

Convert color representation of selected images from RGB to Lch8.

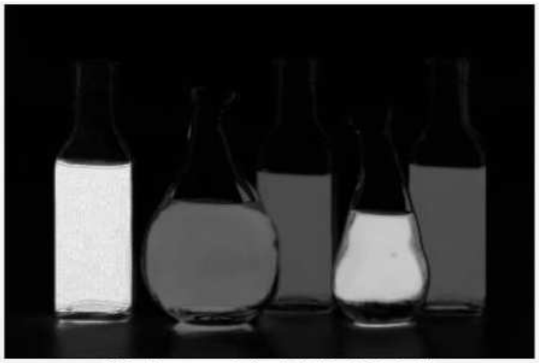
## Default values:

`illuminant=2`.

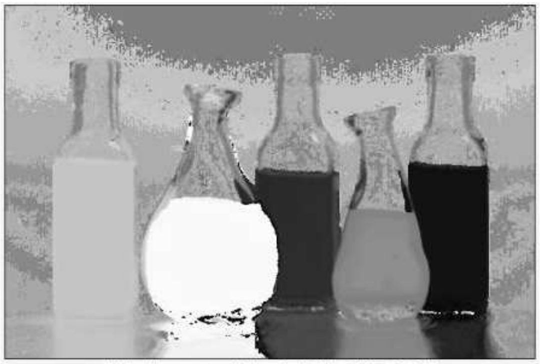
## Example of use:

```
image.jpg rgb2lch8 split c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

---

# rgb2luv

### No arguments

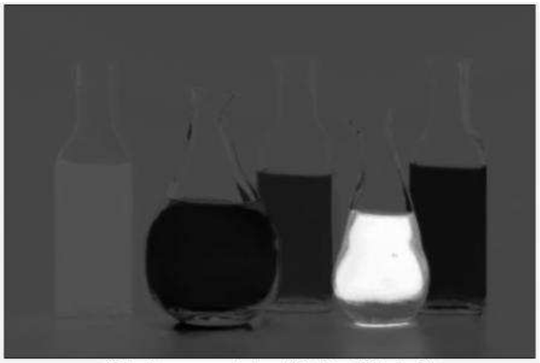
## Description:

Convert color representation of selected images from RGB to LUV.

## Example of use:

```
image.jpg rgb2luv split c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)

[2]: 'image_c2.jpg' (640x427x1x1)

# rgb2oklab

**No arguments**

## Description:

Convert color representation of selected images from RGB to Oklab.

(see colorspace definition at: **https://bottosson.github.io/posts/oklab/** ).

## See also:

**oklab2rgb** .

# rgb2ryb

**No arguments**

## Description:

Convert color representation of selected images from RGB to RYB.

## Example of use:

```
image.jpg rgb2ryb split c
```

[0]: 'image.jpg' (640x427x1x1)

[1]: 'image_c1.jpg' (640x427x1x1)

[2]: 'image_c2.jpg' (640x427x1x1)

## rgb2srgb

**No arguments**

### Description:

Convert color representation of selected images from linear RGB to sRGB.

## rgb2xyz

### Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

### Description:

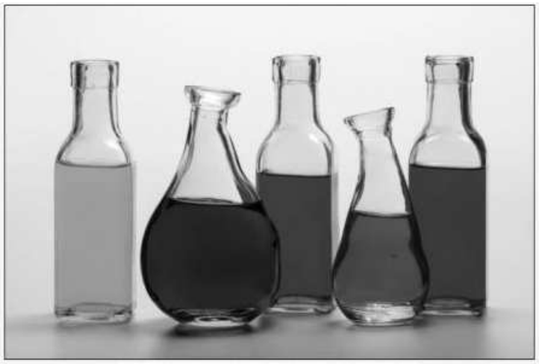
Convert color representation of selected images from RGB to XYZ.

### Default values:

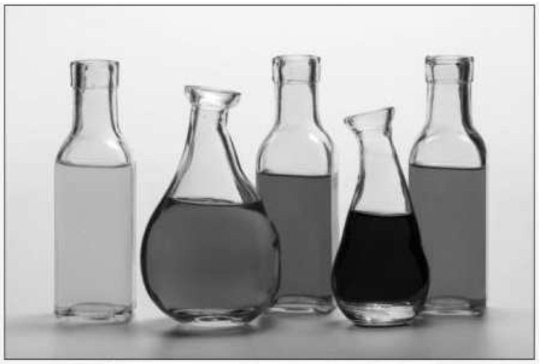
`illuminant=2`.

### Example of use:

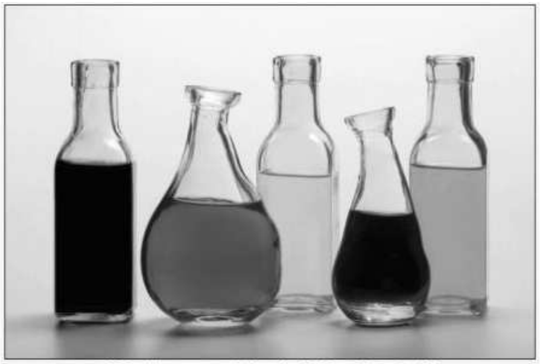
```
image.jpg rgb2xyz split c
```



[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

# rgb2xyz8

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from RGB to XYZ8.

## Default values:

`illuminant=2`.

## Example of use:

```
image.jpg rgb2xyz8 split c
```

[0]: 'image.jpg' (640x427x1x1)

[1]: 'image_c1.jpg' (640x427x1x1)

[2]: 'image_c2.jpg' (640x427x1x1)

# rgb2ycbcr

**No arguments**

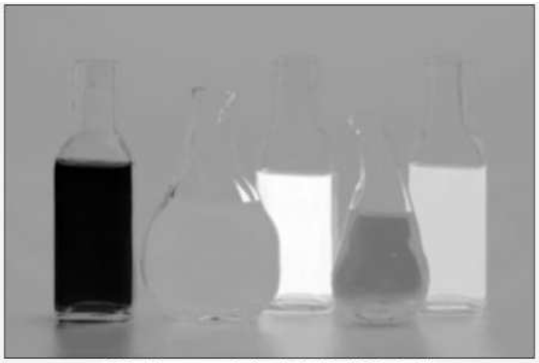
## Description:

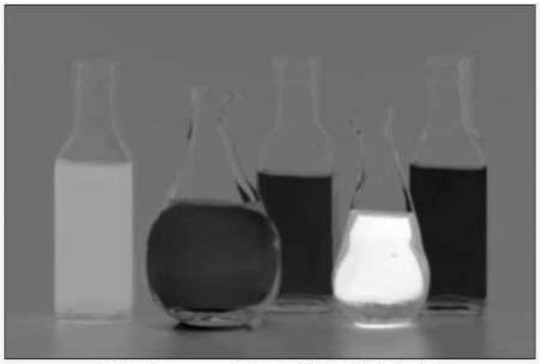
Convert color representation of selected images from RGB to YCbCr.

## Example of use:

```
image.jpg rgb2ycbcr split c
```

[0]: 'image.jpg' (640x427x1x1)


[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

## rgb2yiq

**No arguments**

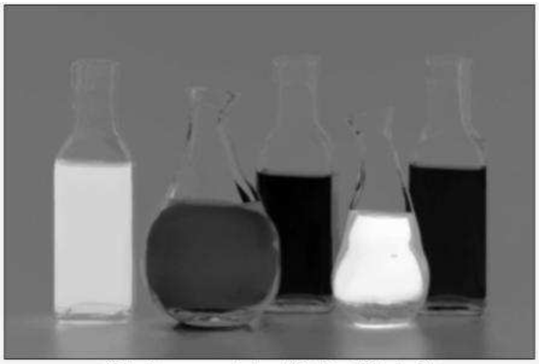
### Description:

Convert color representation of selected images from RGB to YIQ.

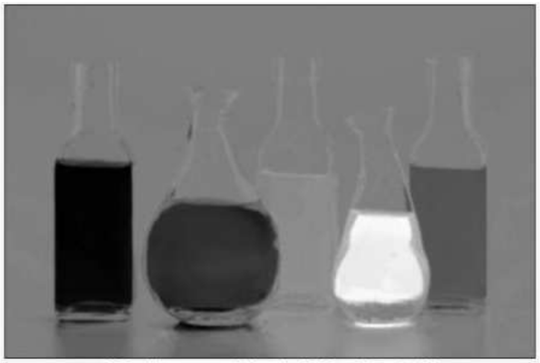
### Example of use:

```
image.jpg rgb2yiq split c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

## rgb2yiq8

**No arguments**

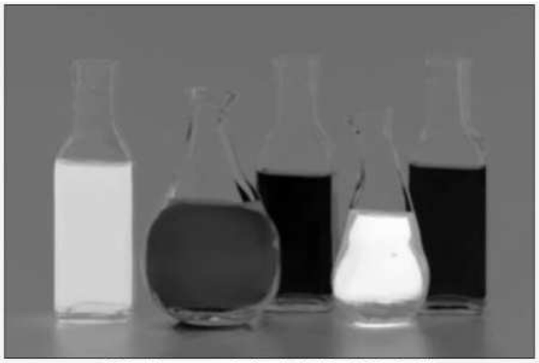
## Description:

Convert color representation of selected images from RGB to YIQ8.

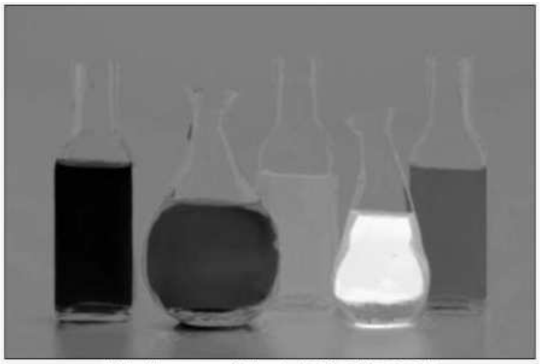
## Example of use:

```
image.jpg rgb2yiq8 split c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

# rgb2yuv

**No arguments**

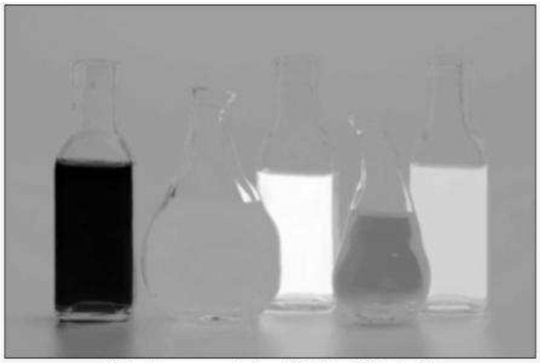
## Description:

Convert color representation of selected images from RGB to YUV.

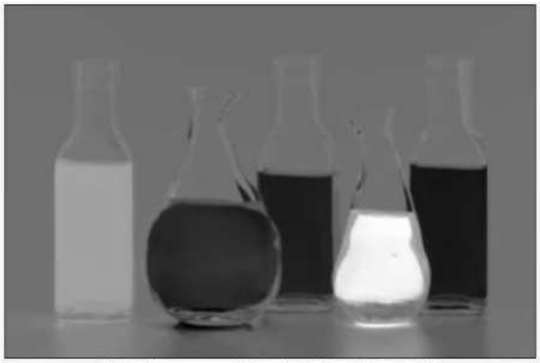
## Example of use:

```
image.jpg rgb2yuv split c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

---

# rgb2yuv8

**No arguments**

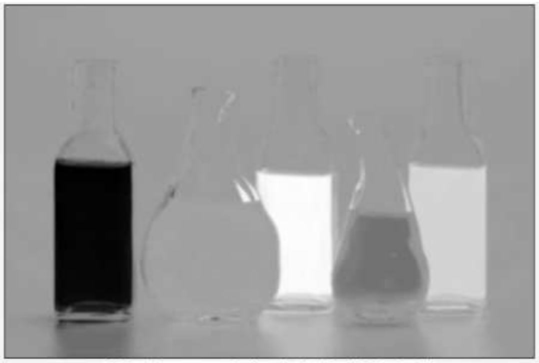
## Description:

Convert color representation of selected images from RGB to YUV8.

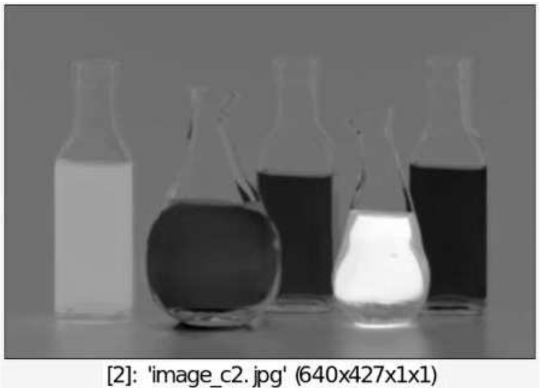
## Example of use:

```
image.jpg rgb2yuv8 split c
```

[0]: 'image.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

# rgba

**No arguments**

## Description:

Return a random int-valued RGBA color.

# ripple

## Arguments:

- `_amplitude,_bandwidth,_shape={ 0:block | 1:triangle | 2:sine | 3:sine+ | 4:random },_angle,_offset`

## Description:

Apply ripple deformation on selected images.

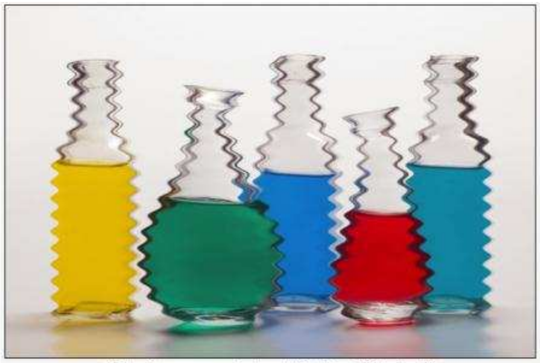
## Default values:

`amplitude=10`, `bandwidth=10`, `shape=2`, `angle=0` and `offset=0`.

## Example of use:

```
image.jpg +ripple ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# rodilius

## Arguments:

- `0<=_amplitude<=100,_0<=thickness<=100,_sharpness>=0,_nb_orientations>0,_offse 0:darker | 1:brighter }`

## Description:

Apply rodilius (fractalius-like) filter on selected images.

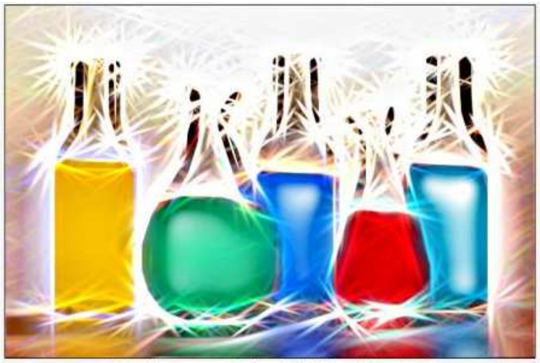
## Default values:

`amplitude=10`, `thickness=10`, `sharpness=400`, `nb_orientations=7`, `offset=0` and `color_mode=1`.

## Examples of use:

### • Example #1

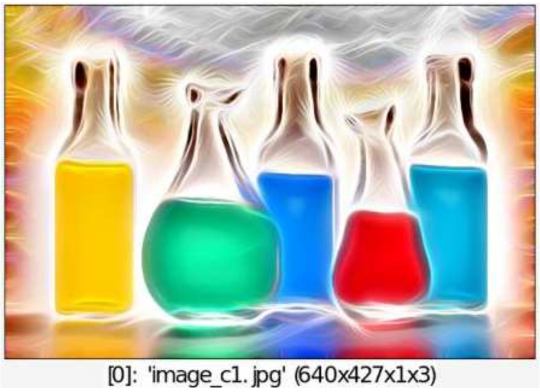
```
image.jpg rodilius 12,10,300,10 normalize_local 10,6
```



[0]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
image.jpg normalize_local 10,16 rodilius 10,4,400,16 smooth
60,0,1,1,4 normalize_local 10,16
```



# rol

Built-in command

## Arguments:

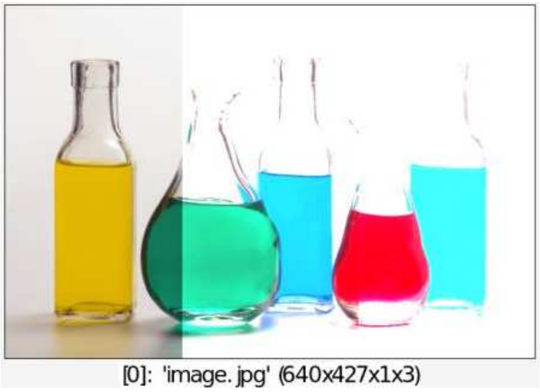
- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

Compute the bitwise left rotation of selected images with specified value, image or mathematical expression, or compute the pointwise sequential bitwise left rotation of selected images.

## Example of use:

```
image.jpg rol 'round(3*x/w,0)' cut 0,255
```



# rolling_guidance

## Arguments:

- `std_deviation_s[%]>=0,std_deviation_r[%]>=0,_precision>=0`

## Description:

Apply the rolling guidance filter on selected image.

Rolling guidance filter is a fast image abstraction filter, described in:
"Rolling Guidance Filter", Qi Zhang Xiaoyong, Shen Li, Xu Jiaya Jia, ECCV'2014.

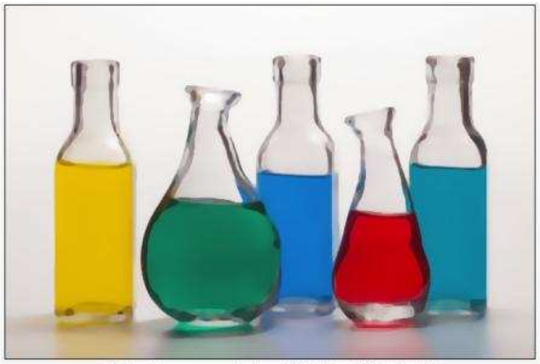
## Default values:

`std_deviation_s=4`, `std_deviation_r=10` and `precision=0.5`.

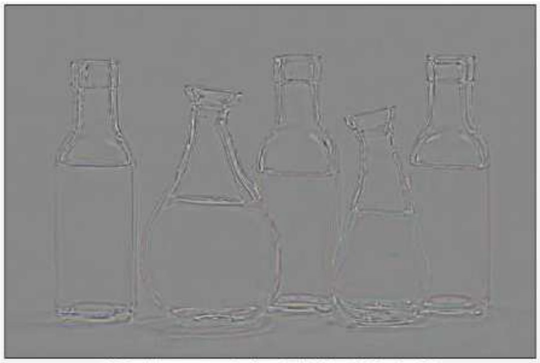
## Example of use:

```
image.jpg +rolling_guidance , +-
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

---

# ror

Built-in command

## Arguments:

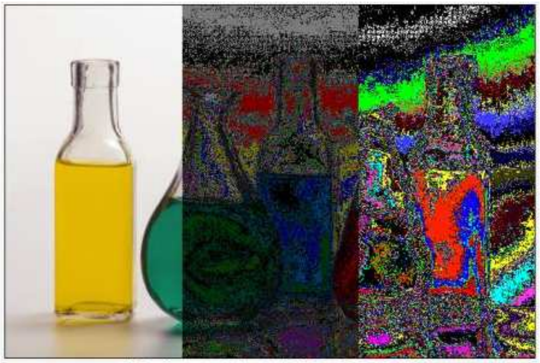
- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

Compute the bitwise right rotation of selected images with specified value, image or mathematical expression, or compute the pointwise sequential bitwise right rotation of selected images.

## Example of use:

```
image.jpg ror 'round(3*x/w,0)' cut 0,255
```



[0]: 'image.jpg' (640x427x1x3)

# rorschach

## Arguments:

- `'smoothness[%]>=0','mirroring={ 0:none | 1:x | 2:y | 3:xy }`

## Description:

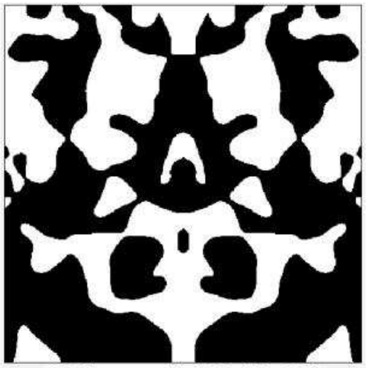
Render rorschach-like inkblots on selected images.

## Default values:

`smoothness=5%` and `mirroring=1`.

## Example of use:

```
400,400 rorschach 3%
```



[0]: '[unnamed]' (400x400x1x1)

# rotate

Built-in command

## Arguments:

- `angle,_interpolation,_boundary_conditions,_center_x[%],_center_y[%]` or
- `u,v,w,angle,interpolation,boundary_conditions,_center_x[%],_center_y[%],_cent`

## Description:

Rotate selected images with specified angle (in deg.), and optionally 3D axis (u,v,w).

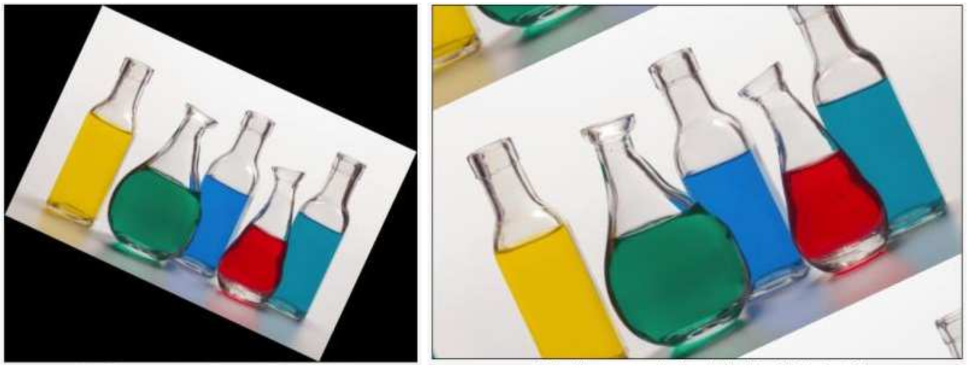
`interpolation` can be ***{ 0:none | 1:linear | 2:bicubic }***.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.
When a rotation center (cx,cy,_cz) is specified, the size of the image is preserved.

## Default values:

`interpolation=1`, `boundary_conditions=0` and `center_x=center_y=(undefined)`.

## Example of use:

```
image.jpg +rotate -25,1,2,50%,50% rotate[0] 25
```



[0]: 'image.jpg' (760x657x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

# rotate3d

Built-in command

## Arguments:

- `u,v,w,angle`

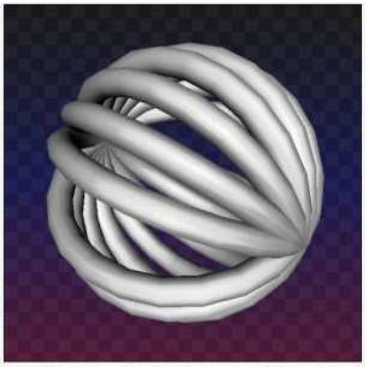
## Description:

Rotate selected 3D objects around specified axis with specified angle (in deg.).

(*equivalent to shortcut command* `r3d`).

## Example of use:

```
torus3d 100,10 double3d 0 repeat 7 { +rotate3d[-1] 1,0,0,20 } add3d
```



[0]: '[3D torus]' (2304 vert., 2304 prim.)

# rotate_tileable

## Arguments:

- `angle,_max_size_factor>=0`

## Description:

Rotate selected images by specified angle and make them tileable.

If resulting size of an image is too big, the image is replaced by a 1x1 image.

## Default values:

`max_size_factor=8`.

# rotate_tiles

## Arguments:

- `angle,_M>0,N>0`

## Description:

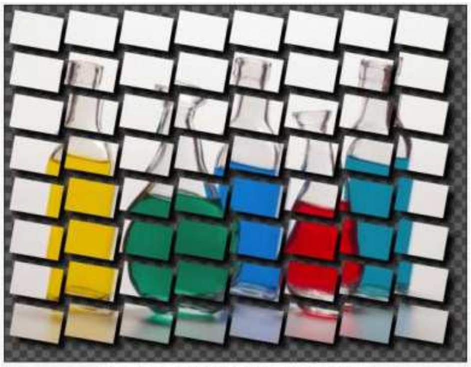
Apply MxN tiled-rotation effect on selected images.

## Default values:

`M=8` and `N=M`.

## Example of use:

```
image.jpg to_rgba rotate_tiles 10,8 drop_shadow 10,10 display_rgba
```



[0]: 'image.jpg' (739x570x1x3)

# rotation3d

## Arguments:

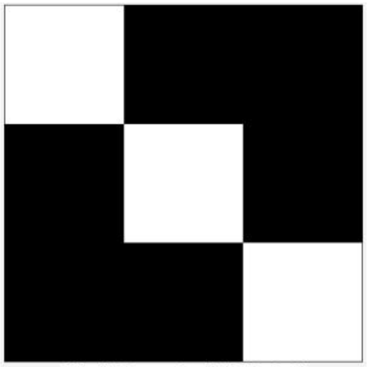
- `u,v,w,angle`

## Description:

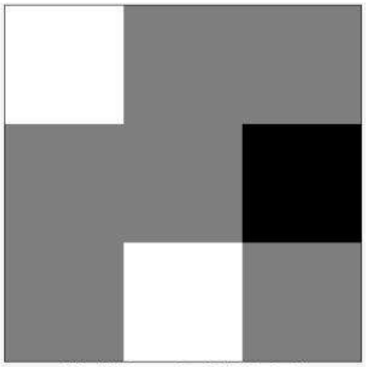
Input 3x3 rotation matrix with specified axis and angle (in deg).

## Example of use:

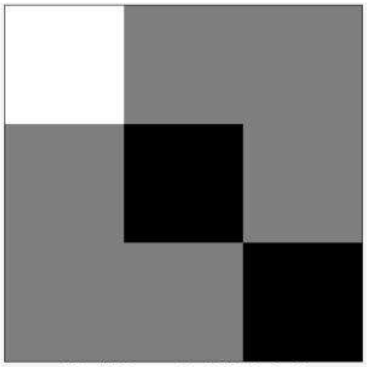
```
rotation3d 1,0,0,0 rotation3d 1,0,0,90 rotation3d 1,0,0,180
```



[0]: '[3D rotation]' (3x3x1x1)



[1]: '[3D rotation]' (3x3x1x1)



[2]: '[3D rotation]' (3x3x1x1)

# rotoidoscope

## Arguments:

- `_center_x[%],_center_y[%],_tiles>0,_smoothness[%]>=0,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Create rotational kaleidoscope effect from selected images.

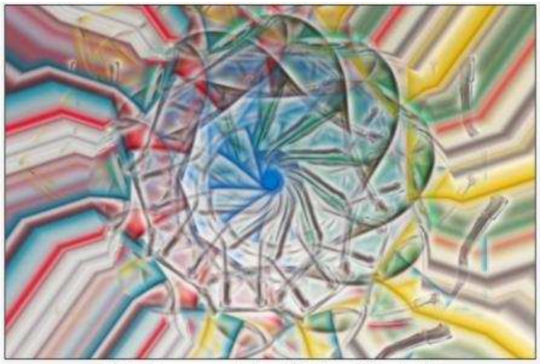
## Default values:

`center_x=center_y=50%`, `tiles=10`, `smoothness=1` and `boundary_conditions=3`.

## Example of use:

```
image.jpg +rotoidoscope ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# round

Built-in command

## Arguments:

- `rounding_value>=0,_rounding_type` or
- `(no arg)`

## Description:

Round values of selected images.

`rounding_type` can be ***{ -1:backward | 0:nearest | 1:forward }***.

## Default values:

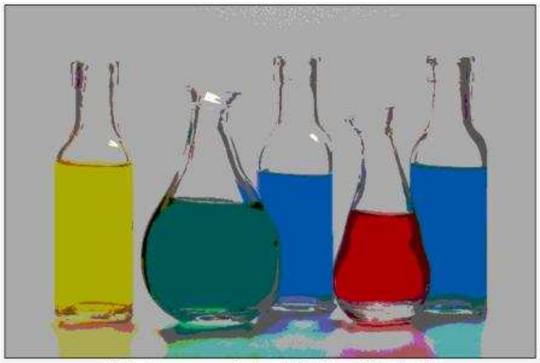
`rounding_type=0`.

## Examples of use:

- **Example #1**

```
image.jpg +round 100
```

[0]: 'image.jpg' (640x427x1x3)

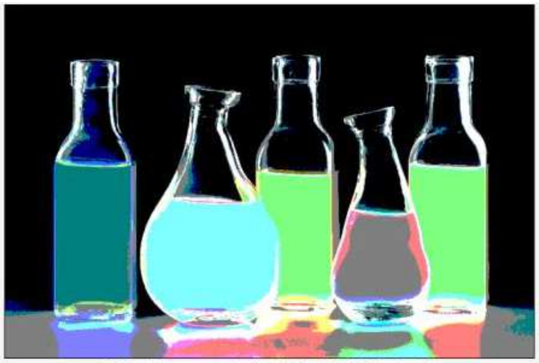


[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg mul {pi/180} sin +round
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# roundify

## Arguments:

- `gamma>=0`

## Description:

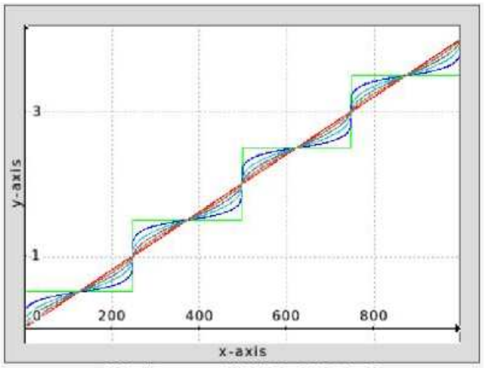
Apply roundify transformation on float-valued data, with specified gamma.

## Default values:

`gamma=0`.

## Example of use:

```
1000 fill '4*x/w' repeat 5 { +roundify[0] {$>*0.2} } append c
display_graph 400,300
```



[0]: '[unnamed]' (400x300x1x3)

---

## rows

### Arguments:

- `y0[%],_y1[%]`

### Description:

Keep only specified rows of selected images.

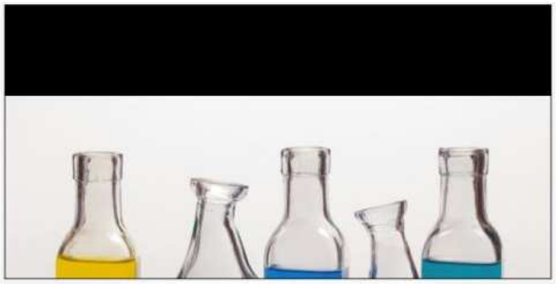
Dirichlet boundary conditions are used when specified rows are out of range.

### Default values:

`y1=y0`.

### Example of use:

```
image.jpg rows -25%,50%
```



[0]: 'image.jpg' (640x320x1x3)

---

## rprogress

### Arguments:

- `0<=value<=100 | -1 | "command",0<=value_min<=100,0<=value_max<=100`

## Description:

Set the progress index of the current processing pipeline (relatively to

previously defined progress bounds), or call the specified command with specified progress bounds.

# run

## Arguments:

- `"G'MIC pipeline"`

## Description:

Run specified G'MIC pipeline.

# ryb2rgb

**No arguments**

## Description:

Convert color representation of selected images from RYB to RGB.

# sample

## Arguments:

- `_name1={ ? | apples | balloons | barbara | boats | bottles | butterfly | cameraman | car | cat | cliff | chick | colorful | david | dog | duck | eagle | elephant | earth | flower | fruits | gmicky | gmicky_mahvin | gmicky_wilber | greece | gummy | house | inside | landscape | leaf | lena | leno | lion | mandrill | monalisa | monkey | parrots | pencils | peppers | portrait0 | portrait1 | portrait2 | portrait3 | portrait4 | portrait5 | portrait6 | portrait7 | portrait8 | portrait9 | roddy | rooster | rose | square | swan | teddy | tiger | tulips | wall | waterfall | zelda },_name2,...,_nameN,_width={ >=0 | 0 (auto) },_height = { >=0 | 0 (auto) }` or
- `(no arg)`

## Description:

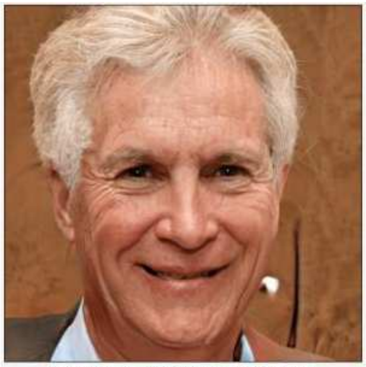
Input a new sample RGB image (opt. with specified size).

(*equivalent to shortcut command* `sp`).

Argument `name` can be replaced by an integer which serves as a sample index.

## Example of use:

```
repeat 6 { sample }
```



[0]: 'portrait0' (800x800x1x3)

[1]: 'cliff' (667x500x1x3)

[2]: 'peppers' (512x512x1x3)

[3]: 'zelda' (763x575x1x3)

[4]: 'rose' (600x500x1x3)

[5]: 'square' (750x500x1x3)

# scale2x

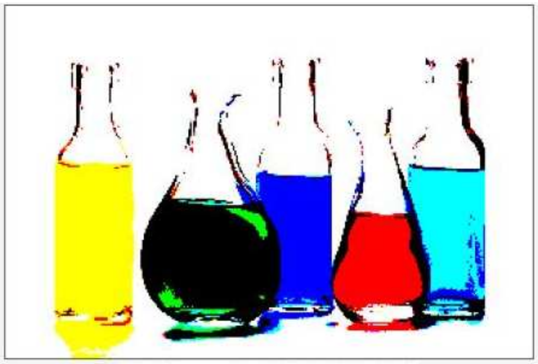
### No arguments

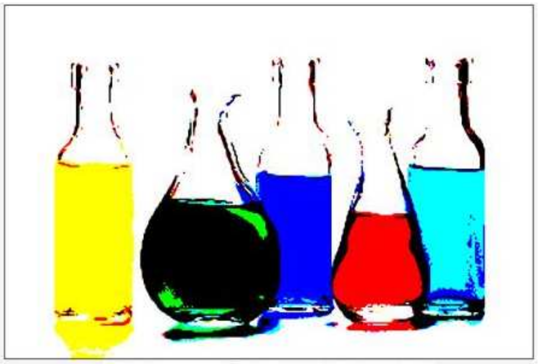
## Description:

Resize selected images using the Scale2x algorithm.

## Example of use:

```
image.jpg threshold 50% resize 50%,50% +scale2x
```


[0]: 'image.jpg' (320x214x1x3)


[1]: 'image_c1.jpg' (640x428x1x3)

# scale3x

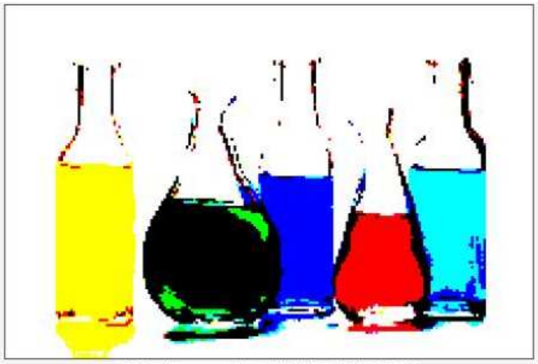
**No arguments**

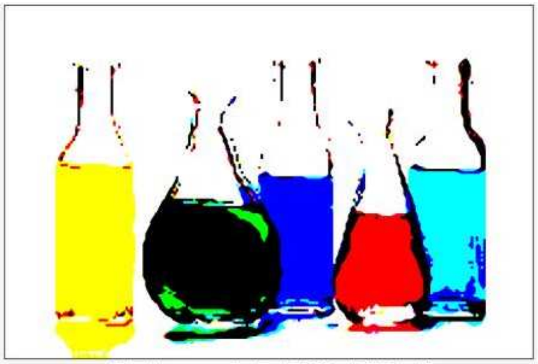
## Description:

Resize selected images using the Scale3x algorithm.

## Example of use:

```
image.jpg threshold 50% resize 33%,33% +scale3x
```


[0]: 'image.jpg' (211x141x1x3)


[1]: 'image_c1.jpg' (633x423x1x3)

# scale_dcci2x

## Arguments:

- `_edge_threshold>=0,_exponent>0,_extend_1px={ 0:false | 1:true }`

## Description:

Double image size using directional cubic convolution interpolation, as described in **https://en.wikipedia.org/wiki/Directional_Cubic_Convolution_Interpolation**.

### Default values:

`edge_threshold=1.15`, `exponent=5` and `extend_1px=0`.

### Example of use:

```
image.jpg +scale_dcci2x ,
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (1279x853x1x3)

## scanlines

### Arguments:

- `_amplitude,_bandwidth,_shape={ 0:block | 1:triangle | 2:sine | 3:sine+ | 4:random },_angle,_offset`

### Description:

Apply ripple deformation on selected images.

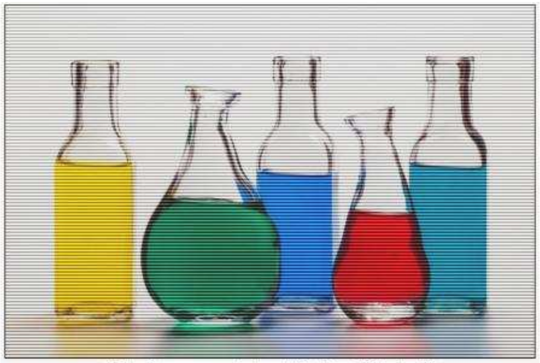
### Default values:

`amplitude=60`, `bandwidth=2`, `shape=0`, `angle=0` and `offset=0`.

### Example of use:

```
image.jpg +scanlines ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## screen

Built-in command

### Arguments:

- `_x0[%],_y0[%],_x1[%],_y1[%]`

### Description:

Take screenshot, optionally grabbed with specified coordinates, and insert it

at the end of the image list.

## seamcarve

### Arguments:

- `_width[%]>=0,_height[%]>=0,_is_priority_channel={ 0 | 1 },_is_antialiasing={ 0 | 1 },_maximum_seams[%]>=0`

### Description:

Resize selected images with specified 2D geometry, using the seam-carving algorithm.

### Default values:

`height=100%`, `is_priority_channel=0`, `is_antialiasing=1` and `maximum_seams=25%`.

### Example of use:

```
image.jpg seamcarve 60%
```

[0]: 'image.jpg' (384x427x1x3)

# segment_watershed

## Arguments:

- `_threshold>=0`

## Description:

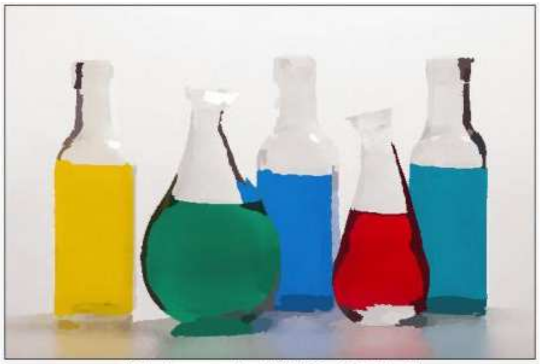
Apply watershed segmentation on selected images.

## Default values:

`threshold=2`.

## Example of use:

```
image.jpg segment_watershed 2
```



[0]: 'image.jpg' (640x427x1x3)

# select

Built-in command

## Arguments:

- `feature_type,_X[%]>=0,_Y[%]>=0,_Z[%]>=0,_exit_on_anykey={ 0 | 1 },_is_deep_selection={ 0 | 1 }`

## Description:

Interactively select a feature from selected images (use the instant display window [0] if opened).

`feature_type` can be ***{ 0:point | 1:segment | 2:rectangle | 3:ellipse }***.
Arguments `X`, `Y`, `Z` determine the initial selection view, for 3D volumetric images.
The retrieved feature is returned as a 3D vector (if `feature_type==0`) or as a 6d vector (if `feature_type!=0`) containing the feature coordinates.

## Default values:

`X=Y=Z=(undefined)`, `exit_on_anykey=0` and `is_deep_selection=0`.

# select_color

## Arguments:

- `tolerance[%]>=0,col1,...,colN`

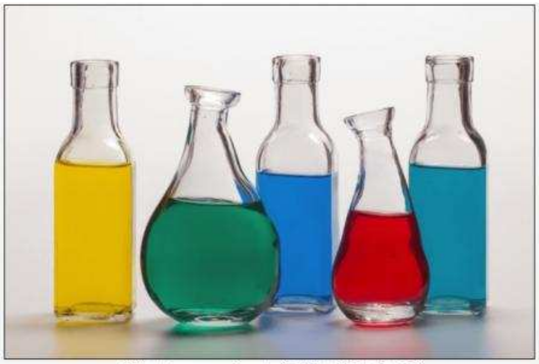
## Description:

Select pixels with specified color in selected images.

This command has a **tutorial page**.

## Example of use:

```
image.jpg +select_color 40,204,153,110
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x1)

# sepia

### No arguments

## Description:

Apply sepia tones effect on selected images.

## Example of use:

```
image.jpg sepia
```

[0]: 'image.jpg' (640x427x1x3)

# serialize

Built-in command

## Arguments:

- `_datatype,_is_compressed={ 0 | 1 },_store_names={ 0 | 1 }`

## Description:

Serialize selected list of images into a single image, optionally in a compressed form.

`datatype` can be ***{ auto | uint8 | int8 | uint16 | int16 | uint32 | int32 | uint64 | int64 | float32 | float64 }***.
Specify `datatype` if all selected images have a range of values constrained to a particular datatype,
in order to minimize the memory footprint.
The resulting image has only integers values in [0,255] and can then be saved as a raw image of unsigned chars (doing so will output a valid .cimg[z] or .gmz file).
If `store_names` is set to `1`, serialization uses the .gmz format to store data in memory (otherwise the .cimg[z] format).

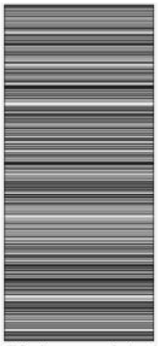
## Default values:

`datatype=auto`, `is_compressed=1` and `store_names=1`.

## Example of use:

```
image.jpg +serialize uint8 +unserialize[-1]
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (1x409951x1x1)

[2]: 'image.jpg' (640x427x1x3)

# set

Built-in command

## Arguments:

- `value,_x[%],_y[%],_z[%],_c[%]`

## Description:

Set pixel value in selected images, at specified coordinates.

(*equivalent to shortcut command* `=`).

If specified coordinates are outside the image bounds, no action is performed.

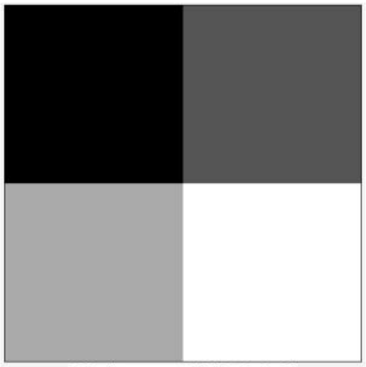
## Default values:

`x=y=z=c=0`.

## Examples of use:

### • Example #1

```
2,2 set 1,0,0 set 2,1,0 set 3,0,1 set 4,1,1
```



[0]: '[unnamed]' (2x2x1x1)

- **Example #2**

```
image.jpg repeat 10000 { set 255,{u(100)}%,{u(100)}%,0,{u(100)}% }
```

[0]: 'image.jpg' (640x427x1x3)

---

# shade_stripes

## Arguments:

- `_frequency>=0,_direction={ 0:horizontal | 1:vertical },_darkness>=0,_lightness>=0`

## Description:

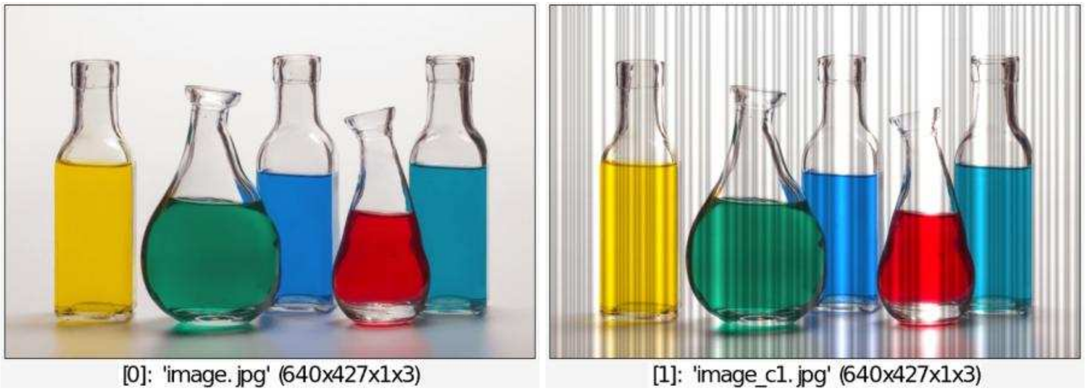
Add shade stripes to selected images.

## Default values:

`frequency=5`, `direction=1`, `darkness=0.8` and `lightness=2`.

## Example of use:

```
image.jpg +shade_stripes 30
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

# shadow_patch

## Arguments:

- `_opacity>=0`

## Description:

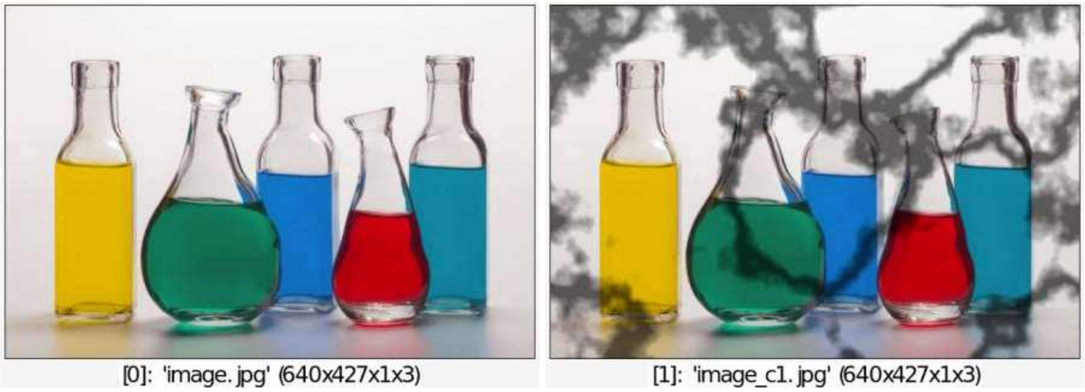
Add shadow patches to selected images.

## Default values:

`opacity=0.7`.

## Example of use:

```
image.jpg +shadow_patch 0.4
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

# shape2bump

## Arguments:

- `_resolution>=0,0<=_weight_std_max_avg<=1,_dilation,_smoothness>=0`

## Description:

Estimate bumpmap from binary shape in selected images.

## Default values:

`resolution=256`, `weight_std_max=0.75`, `dilation=0` and `smoothness=100`.

# shape_circle

## Arguments:

- `_size>=0`

## Description:

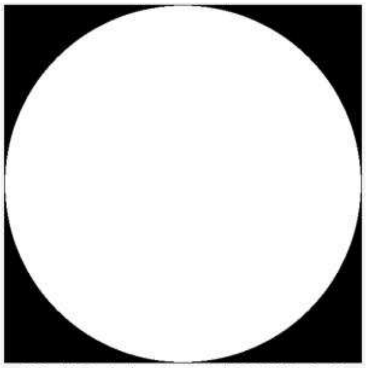
Input a 2D circle binary shape with specified size.

## Default values:

`size=512`.

## Example of use:

```
shape_circle ,
```



[0]: '[2D circle shape]' (512x512x1x1)

# shape_cupid

## Arguments:

- `_size>=0`

## Description:

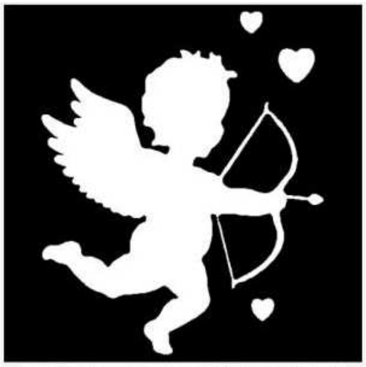
Input a 2D cupid binary shape with specified size.

### Default values:

`size=512`.

### Example of use:

```
shape_cupid ,
```



[0]: '[2D cupid shape]' (512x512x1x1)

## shape_diamond

### Arguments:

- `_size>=0`

### Description:

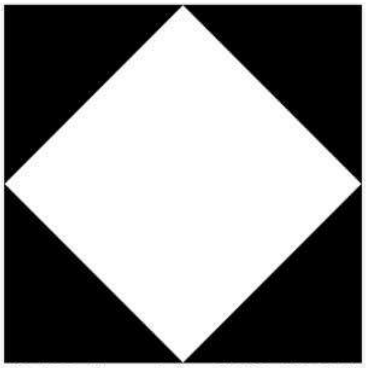
Input a 2D diamond binary shape with specified size.

### Default values:

`size=512`.

### Example of use:

```
shape_diamond ,
```



[0]: '[2D diamond shape]' (512x512x1x1)

---

# shape_dragon

## Arguments:

- `_size>=0,_recursion_level>=0,_angle`

## Description:

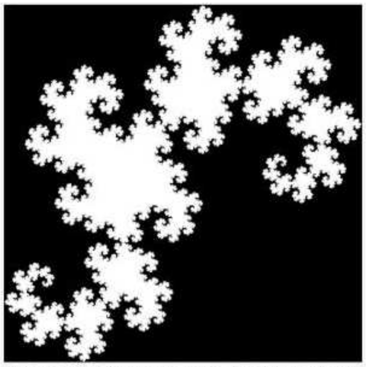
Input a 2D Dragon curve with specified size.

## Default values:

`size=512`, `recursion_level=18` and `angle=0`.

## Example of use:

```
shape_dragon ,
```



[0]: '[2D dragon shape]' (512x512x1x1)

---

# shape_fern

## Arguments:

- `_size>=0,_density[%]>=0,_angle,0<=_opacity<=1,_type={ 0:Asplenium adiantum-nigrum | 1:Thelypteridaceae }`

## Description:

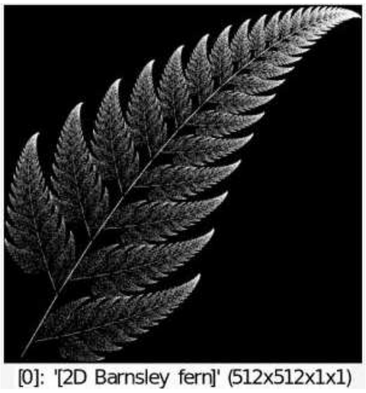
Input a 2D Barnsley fern with specified size.

## Default values:

`size=512`, `density=50%`, `angle=30`, `opacity=0.3` and `type=0`.

## Example of use:

```
shape_fern ,
```


[0]: '[2D Barnsley fern]' (512x512x1x1)

## shape_gear

### Arguments:

- `_size>=0,_nb_teeth>0,0<=_height_teeth<=100,0<=_offset_teeth<=100,0<=_inner_ra`

### Description:

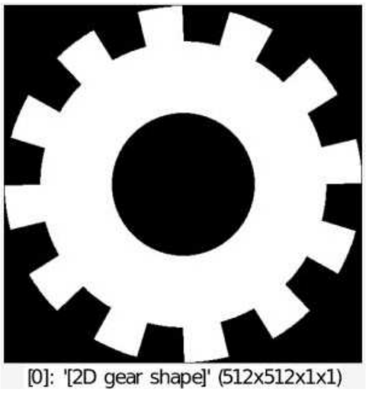
Input a 2D gear binary shape with specified size.

### Default values:

`size=512`, `nb_teeth=12`, `height_teeth=20`, `offset_teeth=0` and `inner_radius=40`.

### Example of use:

```
shape_gear ,
```


[0]: '[2D gear shape]' (512x512x1x1)

## shape_heart

### Arguments:

- `_size>=0`

## Description:

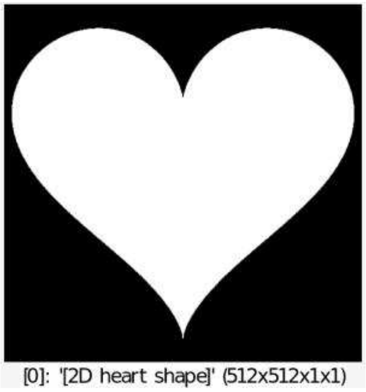
Input a 2D heart binary shape with specified size.

## Default values:

`size=512`.

## Example of use:

```
shape_heart ,
```



[0]: '[2D heart shape]' (512x512x1x1)

# shape_menger

## Arguments:

- `_nb_iterations>=0`

## Description:

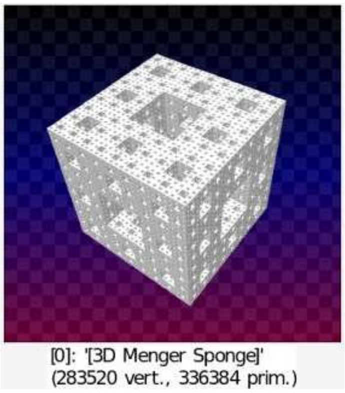
Input a 3D voxelized representation of the Menger sponge.

## Default values:

`nb_iterations=3`.

## Example of use:

```
shape_menger 4 surfels3d , color3d 200 m3d 3
```



[0]: '[3D Menger Sponge]'
(283520 vert., 336384 prim.)

## shape_mosely

### Arguments:

- `_nb_iterations>=0`

### Description:

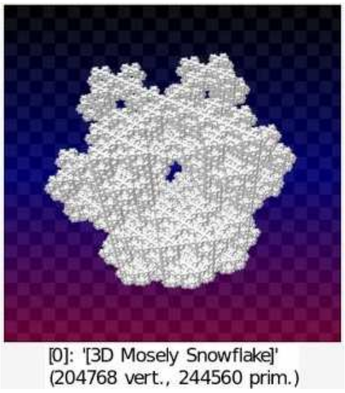
Input a 3D voxelized representation of the Mosely snowflake.

### Default values:

`nb_iterations=3`.

### Example of use:

```
shape_mosely 4 surfels3d , color3d 200 m3d 3
```



[0]: '[3D Mosely Snowflake]'
(204768 vert., 244560 prim.)

## shape_polygon

### Arguments:

- `_size>=0,_nb_vertices>=3,_angle`

## Description:

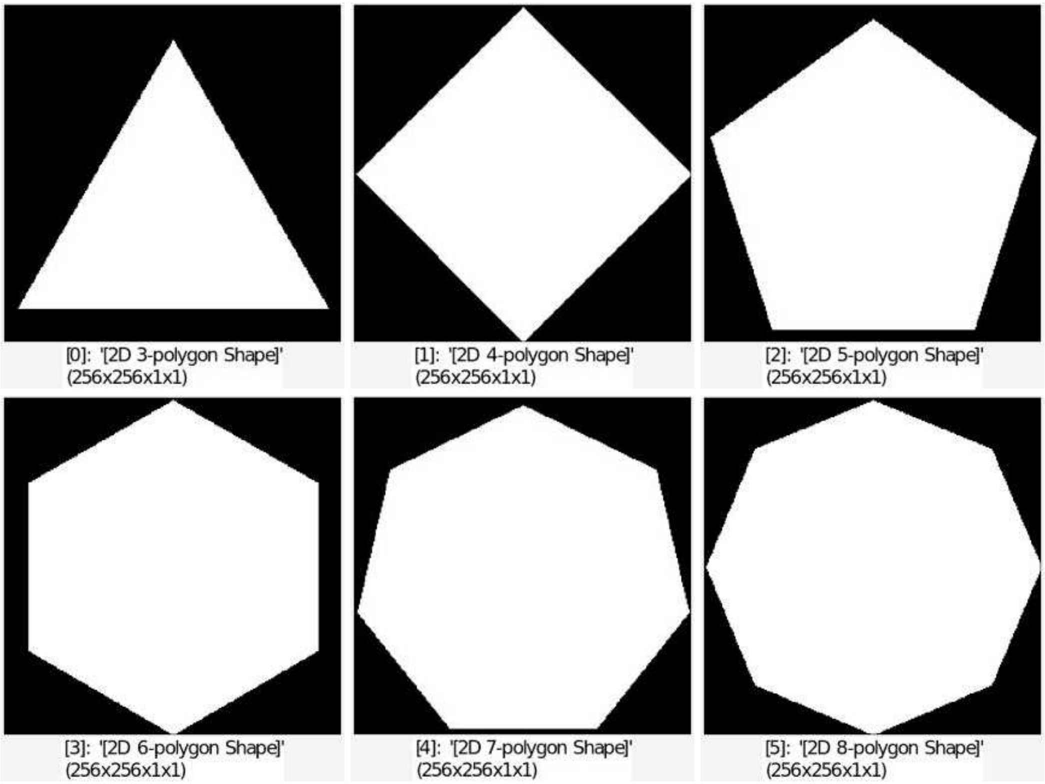
Input a 2D polygonal binary shape with specified geometry.

## Default values:

`size=512`, `nb_vertices=5` and `angle=0`.

## Example of use:

```
repeat 6 { shape_polygon 256,{3+$>} }
```



[0]: '[2D 3-polygon Shape]' (256x256x1x1)

[1]: '[2D 4-polygon Shape]' (256x256x1x1)

[2]: '[2D 5-polygon Shape]' (256x256x1x1)

[3]: '[2D 6-polygon Shape]' (256x256x1x1)

[4]: '[2D 7-polygon Shape]' (256x256x1x1)

[5]: '[2D 8-polygon Shape]' (256x256x1x1)

# shape_rays

## Arguments:

- `_size>=0,_xcenter[%],_ycenter[%],_branches>0,_angle[%],_twist,0<=_perspective<=1,_is_antialias={ 0 | 1 }`

## Description:

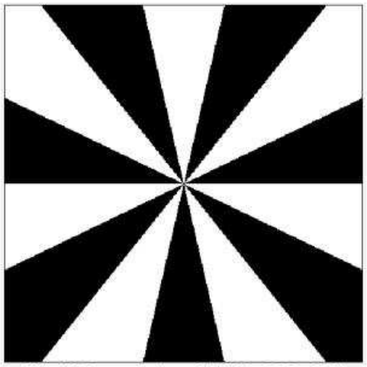
Input a 3D binary spiral with specified size and attributes.

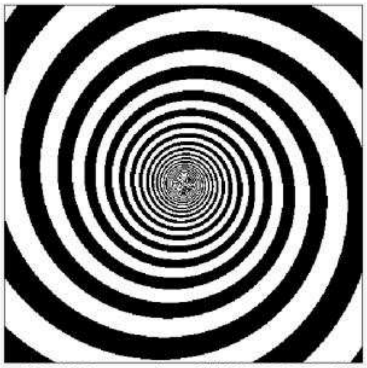
## Default values:

`size=512`, `xcenter=50%`, `ycenter=50%`, `branches=7`, `angle=50%`, `twist=0`, `perspective=0.35` and `is_antialias=0`.

### Example of use:

```
shape_rays 400,50%,50%,7 shape_rays 400,50%,50%,3,0,3
```


[0]: '[2D rays shape]' (400x400x1x1)


[1]: '[2D rays shape]' (400x400x1x1)

---

## shape_snowflake

### Arguments:

- `size>=0,0<=_nb_recursions<=6`

### Description:

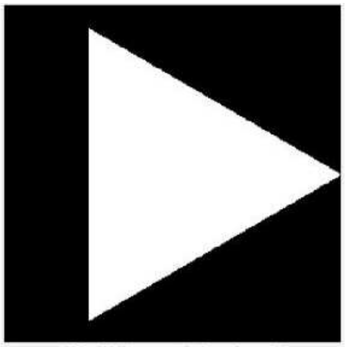
Input a 2D snowflake binary shape with specified size.

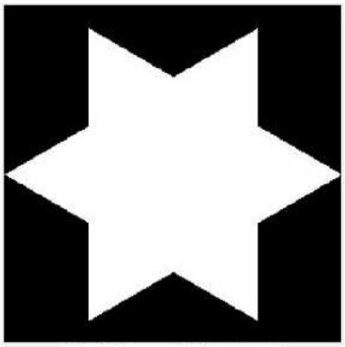
### Default values:

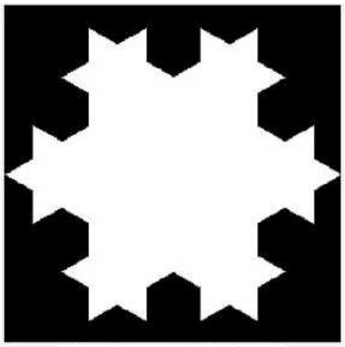
`size=512` and `nb_recursions=5`.

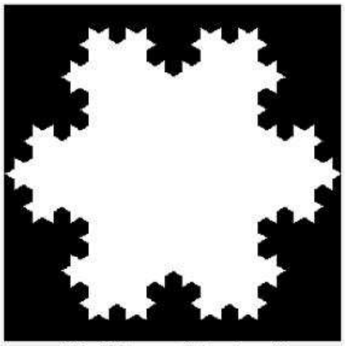
### Example of use:

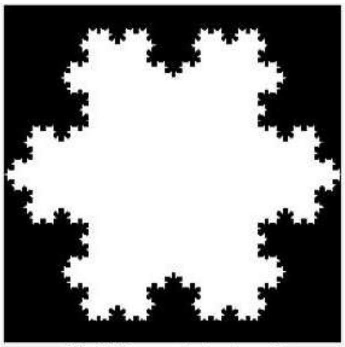
```
repeat 6 { shape_snowflake 256,$> }
```

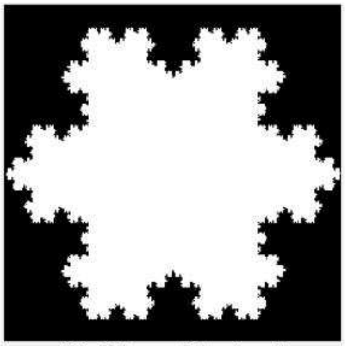

[0]: '[2D snowflake shape]' (256x256x1x1)


[1]: '[2D snowflake shape]' (256x256x1x1)


[2]: '[2D snowflake shape]' (256x256x1x1)


[3]: '[2D snowflake shape]' (256x256x1x1)


[4]: '[2D snowflake shape]' (256x256x1x1)


[5]: '[2D snowflake shape]' (256x256x1x1)

# shape_star

## Arguments:

- `_size>=0,_nb_branches>0,0<=_thickness<=1`

## Description:

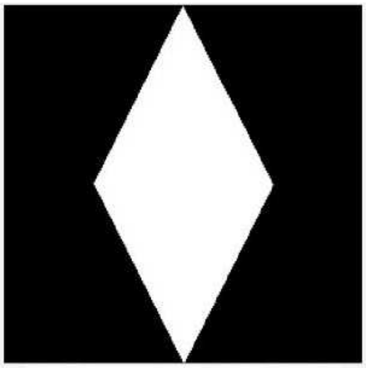
Input a 2D star binary shape with specified size.

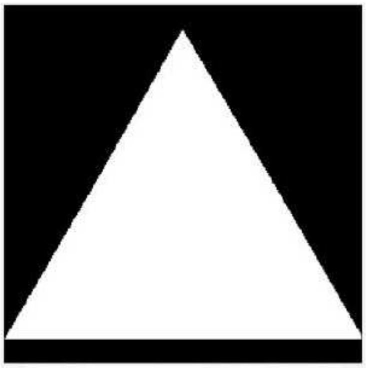
## Default values:

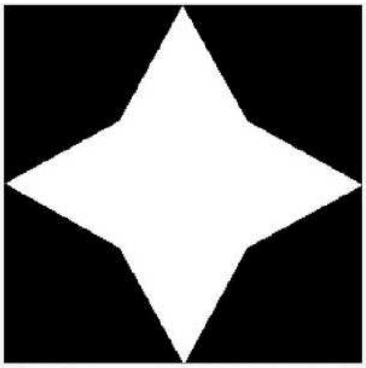
`size=512`, `nb_branches=5` and `thickness=0.38`.

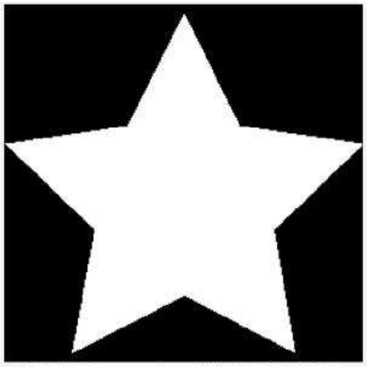
## Example of use:

```
repeat 9 { shape_star 256,{$>+2} }
```

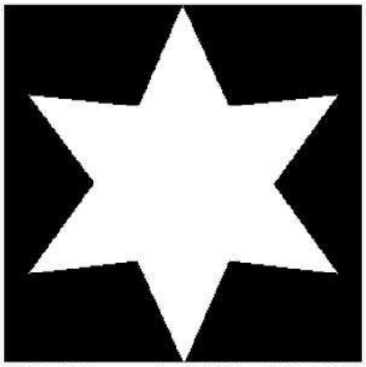

[0]: '[2D star shape]' (256x256x1x1)


[1]: '[2D star shape]' (256x256x1x1)

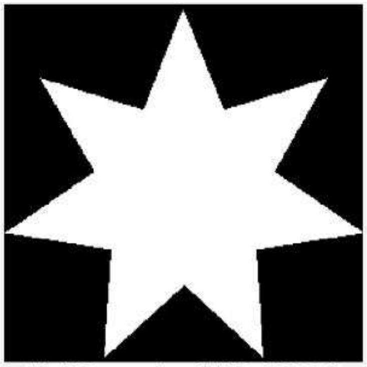

[2]: '[2D star shape]' (256x256x1x1)



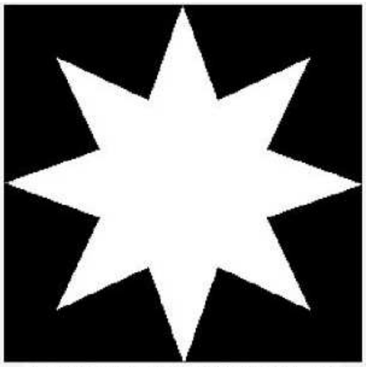
[3]: '[2D star shape]' (256x256x1x1)



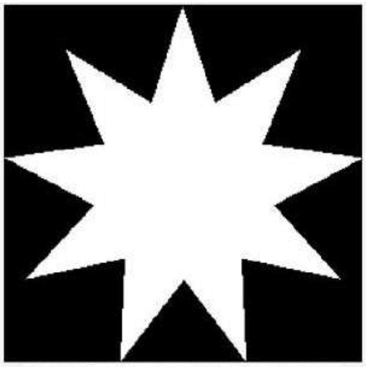
[4]: '[2D star shape]' (256x256x1x1)



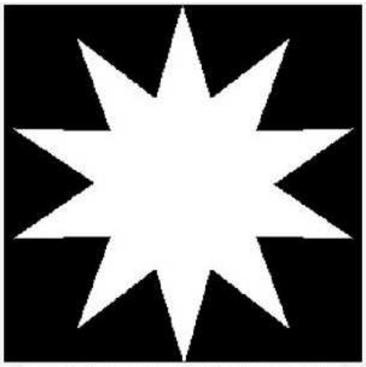
[5]: '[2D star shape]' (256x256x1x1)



[6]: '[2D star shape]' (256x256x1x1)



[7]: '[2D star shape]' (256x256x1x1)



[8]: '[2D star shape]' (256x256x1x1)

# shared

Built-in command

## Arguments:

- `x0[%],x1[%],y[%],z[%],c[%]` or
- `y0[%],y1[%],z[%],c[%]` or
- `z0[%],z1[%],c[%]` or
- `c0[%],c1[%]` or
- `c0[%]` or
- `(no arg)`

## Description:

Insert shared buffers from (opt. points/rows/planes/channels of) selected images.

Shared buffers cannot be returned by a command, nor a local environment.

(*equivalent to shortcut command* `sh`).

This command has a **tutorial page**.

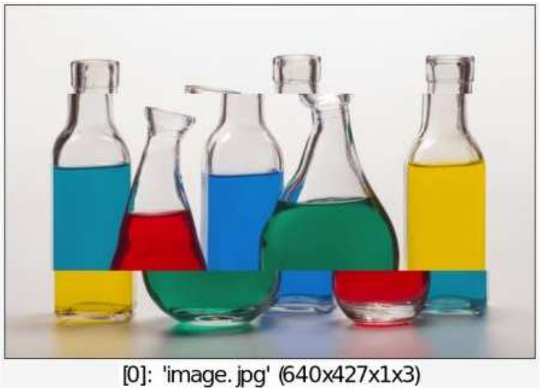
## Examples of use:

### • Example #1

```
image.jpg shared 1 blur[-1] 3 remove[-1]
```

- **Example #2**

```
image.jpg repeat s { shared 25%,75%,0,$> mirror[-1] x remove[-1] }
```



# sharpen

## Arguments:

- `amplitude>=0` or
- `amplitude>=0,edge>=0,_alpha[%],_sigma[%]`

## Description:

Sharpen selected images by inverse diffusion or shock filters methods.

`edge` must be specified to enable shock-filter method.

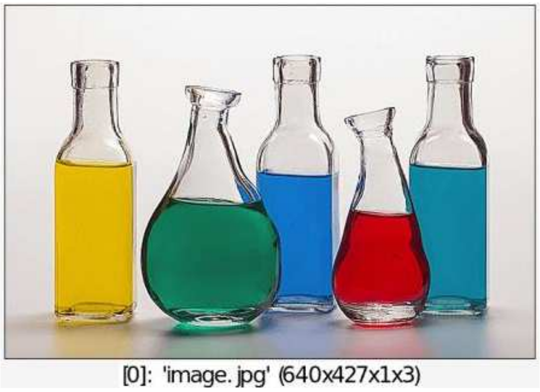
## Default values:

`edge=0`, `alpha=0` and `sigma=0`.

## Examples of use:

- **Example #1**

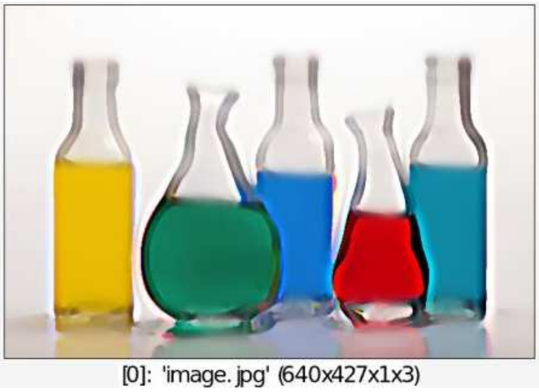
```
image.jpg sharpen 300
```



[0]: 'image.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg blur 5 sharpen 300,1
```



[0]: 'image.jpg' (640x427x1x3)

---

# sharpen_alpha

## Arguments:

- `_amplitude[%]>=0,_nb_scales>0,0<=_anisotropy<=1,0<=_minimize_alpha<=1`

## Description:

Sharpen selected images using a multi-scale and alpha boosting algorithm.

## Default values:

`amplitude=1`, `nb_scales=5`, `anisotropy=0` and `minimize_alpha=1`.

---

# shell_cols

### No arguments

### Description:

Return the estimated number of columns of the current shell.

## shift

Built-in command

### Arguments:

- `vx[%],_vy[%],_vz[%],_vc[%],_boundary_conditions,_interpolation={ 0:nearest_neighbor | 1:linear }`

### Description:

Shift selected images by specified displacement vector.

Displacement vector can be non-integer in which case linear interpolation should be chosen.
`boundary_conditions` can be *{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }*.

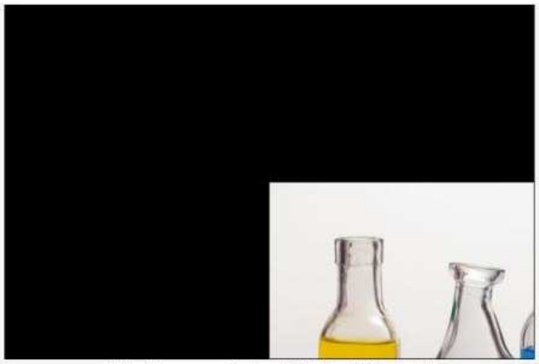
### Default values:

`boundary_conditions=0` and `interpolation=0`.

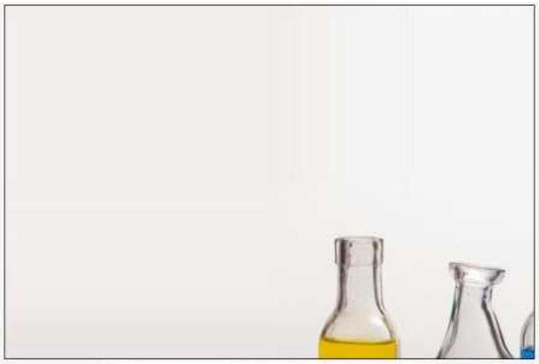
### Example of use:

```
image.jpg +shift[0] 50%,50%,0,0,0 +shift[0] 50%,50%,0,0,1 +shift[0]
50%,50%,0,0,2
```

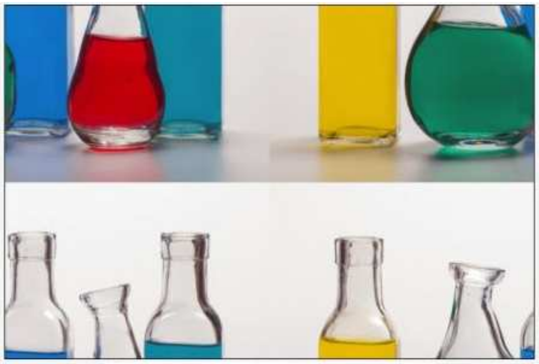
[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)



[3]: 'image_c1.jpg' (640x427x1x3)

# shift_tiles

## Arguments:

- `M>0,_N>0,_amplitude`

## Description:

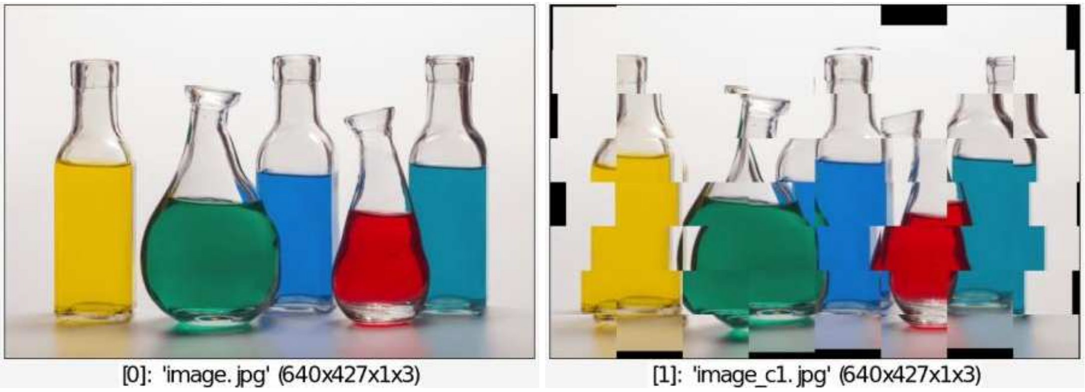
Apply MxN tiled-shift effect on selected images.

## Default values:

`N=M` and `amplitude=20`.

## Example of use:

```
image.jpg +shift_tiles 8,8,10
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

# shrink_x

## Arguments:

- `size_x>=0`

## Description:

Shrink selected images along the x-axis.

## Example of use:

```
image.jpg shrink_x 30
```

[0]: 'image.jpg' (580x427x1x3)

## shrink_xy

### Arguments:

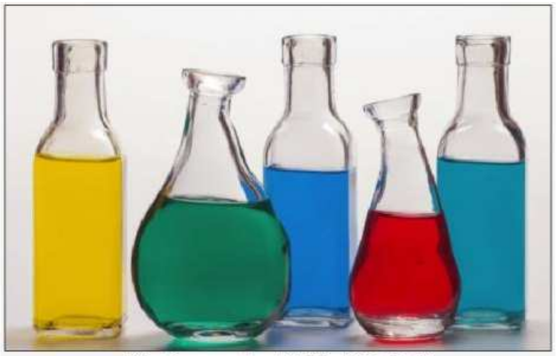
- `size>=0`

### Description:

Shrink selected images along the xy-axes.

### Example of use:

```
image.jpg shrink_xy 30
```



[0]: 'image.jpg' (580x367x1x3)

## shrink_xyz

### Arguments:

- `size>=0`

### Description:

Shrink selected images along the xyz-axes.

# shrink_y

## Arguments:

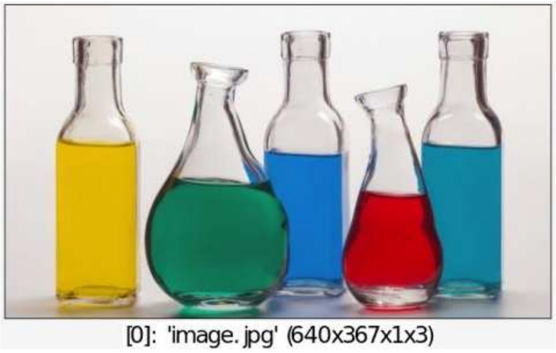
- `size_y>=0`

## Description:

Shrink selected images along the y-axis.

## Example of use:

```
image.jpg shrink_y 30
```



[0]: 'image.jpg' (640x367x1x3)

# shrink_z

## Arguments:

- `size_z>=0`

## Description:

Shrink selected images along the z-axis.

# sierpinski

## Arguments:

- `recursion_level>=0`

## Description:

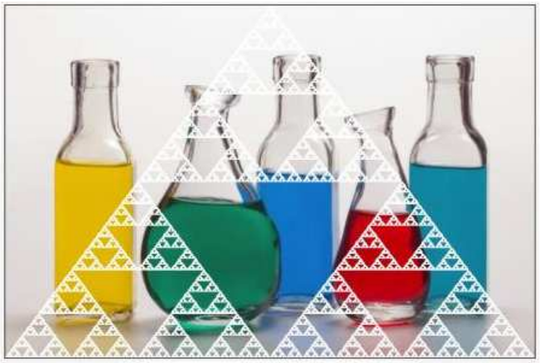
Draw Sierpinski triangle on selected images.

### Default values:

`recursion_level=7`.

### Example of use:

```
image.jpg sierpinski 7
```



[0]: 'image.jpg' (640x427x1x3)

---

## sierpinski3d

### Arguments:

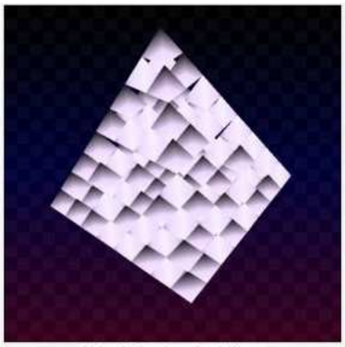
- `_recursion_level>=0,_width,_height`

### Description:

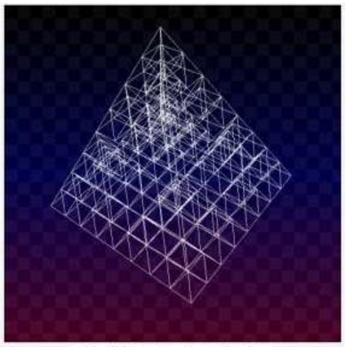
Input 3D Sierpinski pyramid.

### Example of use:

```
sierpinski3d 3,100,-100 +primitives3d 1 color3d[-2] ${-rgb}
```



[0]: '[3D sierpinski]'
(625 vert., 625 prim.)



[1]: '[3D sierpinski]_c1'
(625 vert., 1000 prim.)

---

## sign

Built-in command

### No arguments

## Description:

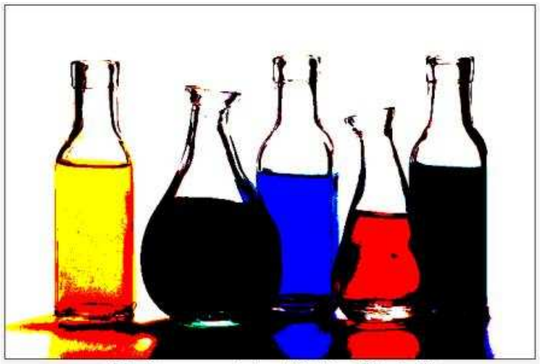
Compute the pointwise sign of selected images.

## Examples of use:

### • Example #1

```
image.jpg +sub {ia} sign[-1]
```

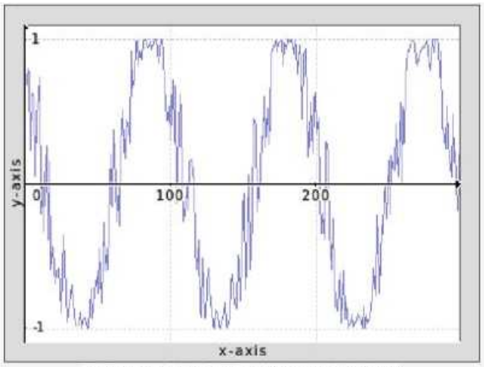
[0]: 'image.jpg' (640x427x1x3)



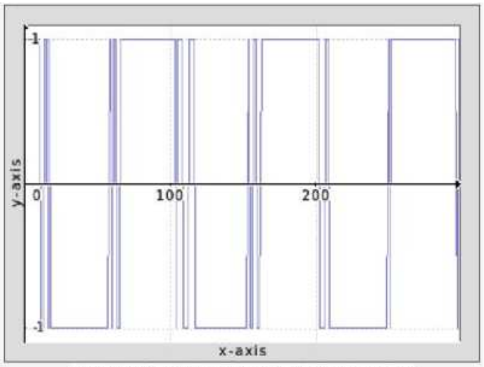
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'cos(20*x/w+u)' +sign display_graph 400,300
```



[0]: '[cos(20*x/w+u)]' (400x300x1x3)



[1]: '[cos(20*x/w+u)]_c1' (400x300x1x3)

---

# sin

**Built-in command**

### No arguments

## Description:

Compute the pointwise sine of selected images.

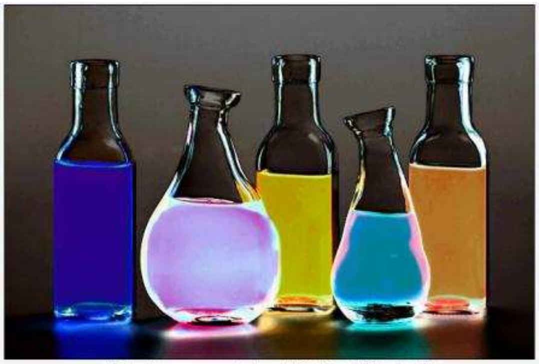
This command has a **tutorial page**.

## Examples of use:

### • Example #1

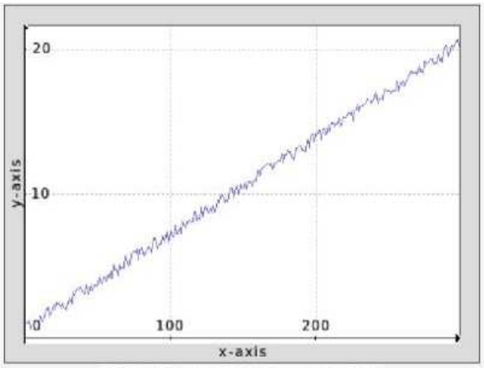
```
image.jpg +normalize 0,{2*pi} sin[-1]
```

[0]: 'image.jpg' (640x427x1x3)

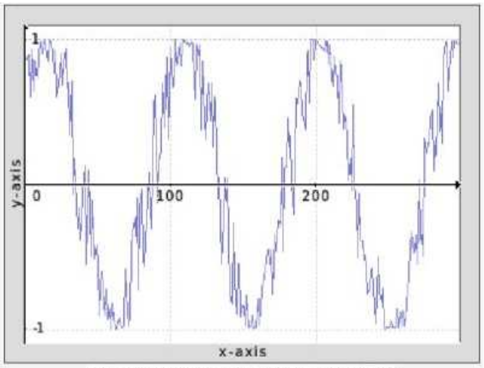

[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'20*x/w+u' +sin display_graph 400,300
```



[0]: '[20*x/w+u]' (400x300x1x3)



[1]: '[20*x/w+u]_c1' (400x300x1x3)

# sinc

Built-in command

### No arguments

## Description:

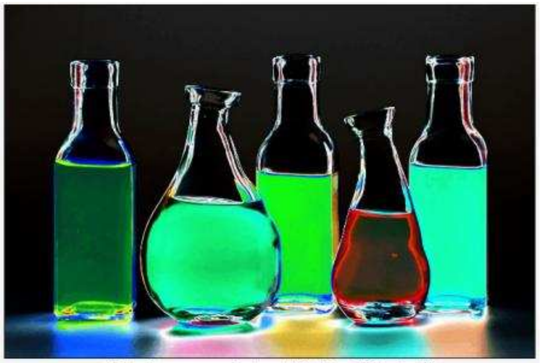
Compute the pointwise sinc function of selected images.

## Examples of use:

### • Example #1

```
image.jpg +normalize {-2*pi},{2*pi} sinc[-1]
```

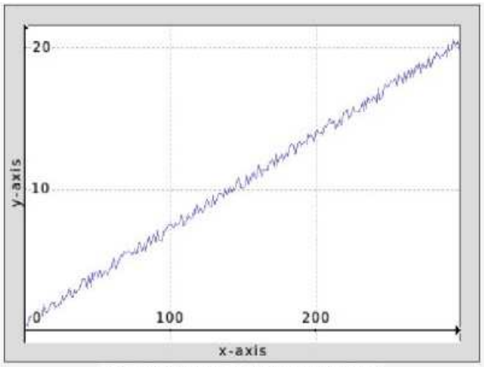
[0]: 'image.jpg' (640x427x1x3)



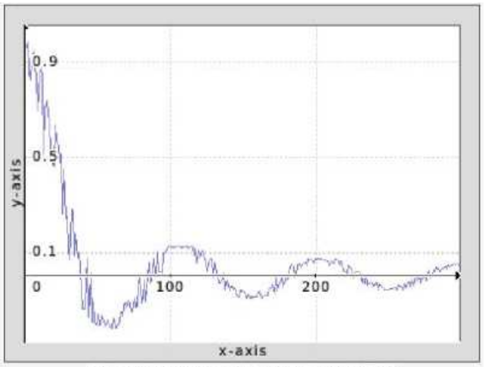
[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
300,1,1,1,'20*x/w+u' +sinc display_graph 400,300
```



[0]: '[20*x/w+u]' (400x300x1x3)



[1]: '[20*x/w+u]_c1' (400x300x1x3)

# sinh

**Built-in command**

**No arguments**

## Description:

Compute the pointwise hyperbolic sine of selected images.

## Examples of use:

- **Example #1**

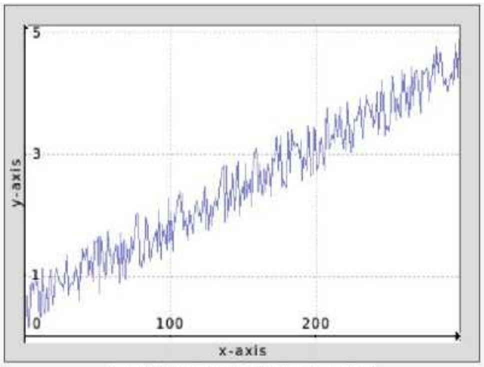
```
image.jpg +normalize -3,3 sinh[-1]
```

[0]: 'image.jpg' (640x427x1x3)

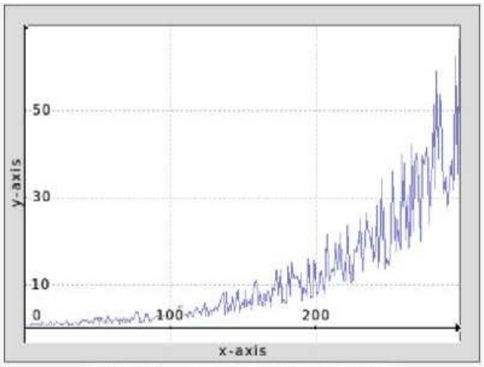
[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
300,1,1,1,'4*x/w+u' +sinh display_graph 400,300
```



[0]: '[4*x/w+u]' (400x300x1x3)



[1]: '[4*x/w+u]_c1' (400x300x1x3)

# size3d

**No arguments**

## Description:

Return bounding box size of the last selected 3D object.

# size_value

**No arguments**

## Description:

Return the size (in bytes) of image values.

# skeleton

## Arguments:

- `_boundary_conditions={ 0:dirichlet | 1:neumann }`

## Description:

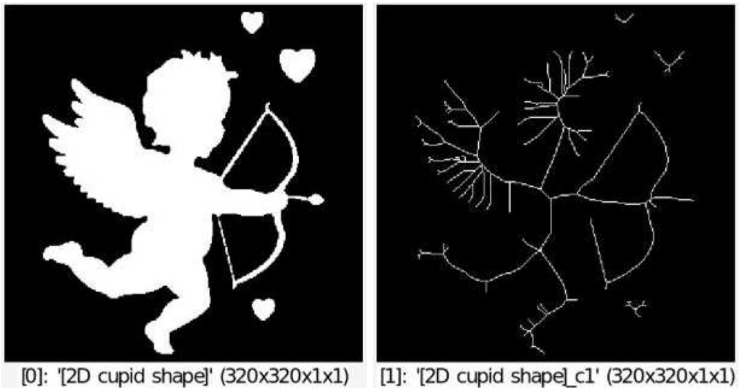
Compute skeleton of binary shapes using distance transform and constrained thinning.

## Default values:

`boundary_conditions=1`.

## Example of use:

```
shape_cupid 320 +skeleton 0
```



[0]: '[2D cupid shape]' (320x320x1x1) [1]: '[2D cupid shape]_c1' (320x320x1x1)

# skeleton3d

## Arguments:

- `_metric,_frame_type={ 0:squares | 1:diamonds | 2:circles | 3:auto },_skeleton_opacity,_frame_opacity,_is_frame_wireframe={ 0 | 1 }`

## Description:

Build 3D skeletal structure object from 2d binary shapes located in selected images.

`metric` can be ***{ 0:chebyshev | 1:manhattan | 2:euclidean }***.

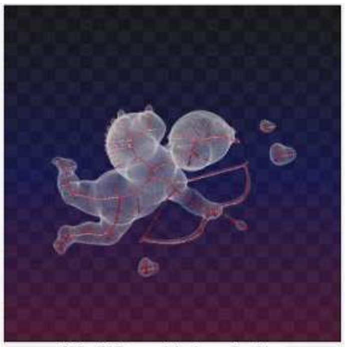
## Default values:

`metric=2`, `bones_type=3`, `skeleton_opacity=1` and `frame_opacity=0.1`.

## Example of use:

```
shape_cupid 480 +skeleton3d ,
```

[0]: '[2D cupid shape]' (480x480x1x1)



[1]: '[2D cupid shape]_c1'
(19161 vert., 19031 prim.)

# sketchbw

## Arguments:

- `_nb_angles>0,_start_angle,_angle_range>=0,_length>=0,_threshold>=0,_opacity,_`
  `0 | 1 },_is_curved={ 0 | 1 }`

## Description:

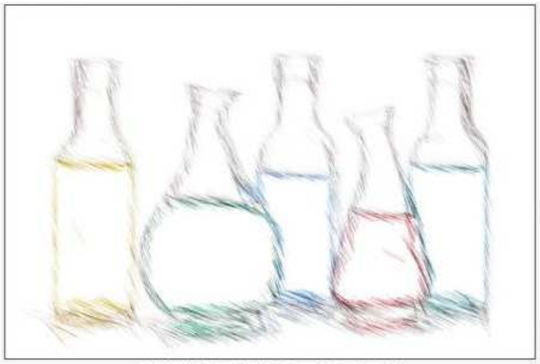
Apply sketch effect to selected images.

## Default values:

`nb_angles=2`, `start_angle=45`, `angle_range=180`, `length=30`, `threshold=3`, `opacity=0.03`, `bgfactor=0`, `density=0.6`, `sharpness=0.1`, `anisotropy=0.6`, `smoothness=0.25`, `coherence=1`, `is_boost=0` and `is_curved=1`.

## Example of use:

```
image.jpg +sketchbw 1 reverse blur[-1] 3 blend[-2,-1] overlay
```



[0]: '[0]_c1' (640x427x1x3)

# skip

**Built-in command**

## Arguments:

- `item`

## Description:

Do nothing but skip specified item.

# slic

## Arguments:

- `size>0,_regularity>=0,_nb_iterations>0`

## Description:

Segment selected 2D images with superpixels, using the SLIC algorithm (Simple Linear Iterative Clustering).

Scalar images of increasingly labeled pixels are returned.
Reference paper: Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., & Susstrunk, S. (2010). SLIC Superpixels (No. EPFL-REPORT-149300).

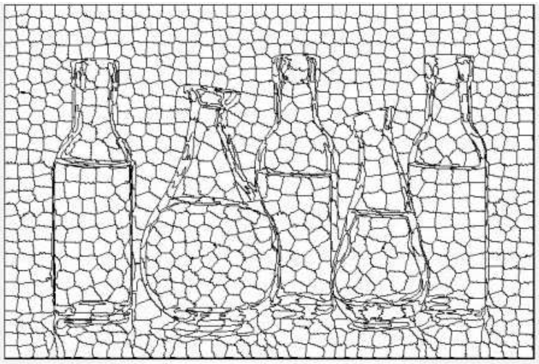
## Default values:

`size=16`, `regularity=10` and `nb_iterations=10`.

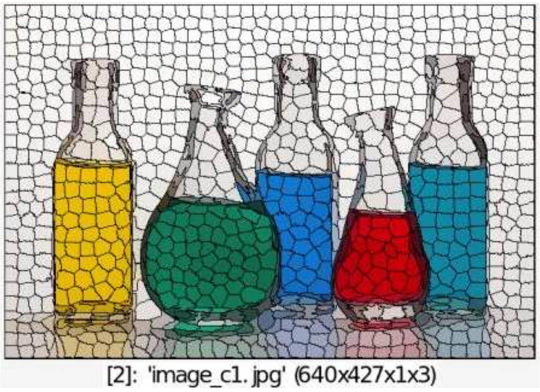
## Example of use:

```
image.jpg +srgb2lab slic[-1] 16 +blend shapeaverage f[-2] "j(1,0)==i
&& j(0,1)==i" *[-1] [-2]
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[unnamed]' (640x427x1x1)



[2]: 'image_c1.jpg' (640x427x1x3)

---

# slices

## Arguments:

- `z0[%],_z1[%]`

## Description:

Keep only specified slices of selected images.

Dirichlet boundary conditions are used when specified slices are out of range.

## Default values:

`z1=z0`.

---

# smooth

**Built-in command**

## Arguments:

- `amplitude[%]>=0,_sharpness>=0,0<=_anisotropy<=1,_alpha[%],_sigma[%],_dl>0,_da`
  `0 | 1 }` or
- `nb_iterations>=0,_sharpness>=0,_anisotropy,_alpha,_sigma,_dt>0,0` or
- `[tensor_field],_amplitude>=0,_dl>0,_da>0,_precision>0,_interpolation,_fast_ap`
  `0 | 1 }` or
- `[tensor_field],_nb_iters>=0,_dt>0,0`

## Description:

Smooth selected images anisotropically using diffusion PDE's, with specified field of

diffusion tensors.
`interpolation` can be ***{ 0:nearest | 1:linear | 2:runge-kutta }***.

## Default values:

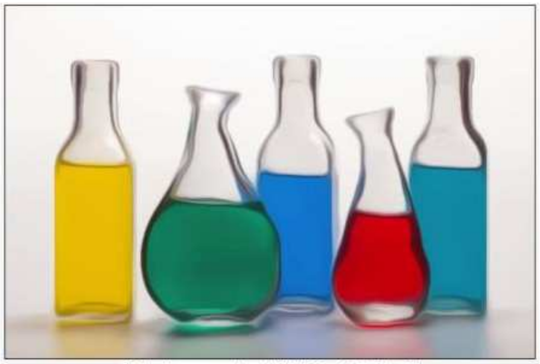
`sharpness=0.7`, `anisotropy=0.3`, `alpha=0.6`, `sigma=1.1`, `dl=0.8`, `da=30`, `precision=2`, `interpolation=0` and `fast_approx=1`.

This command has a **tutorial page**.

## Examples of use:

### • Example #1

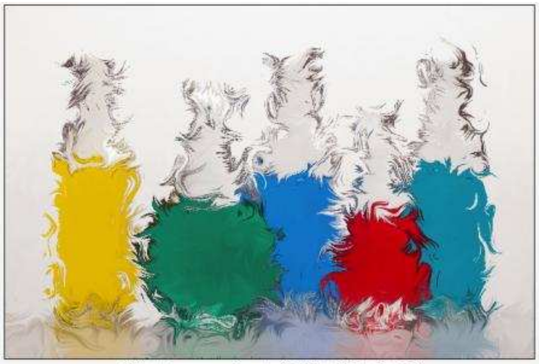
```
image.jpg repeat 3 smooth 40,0,1,1,2 done
```



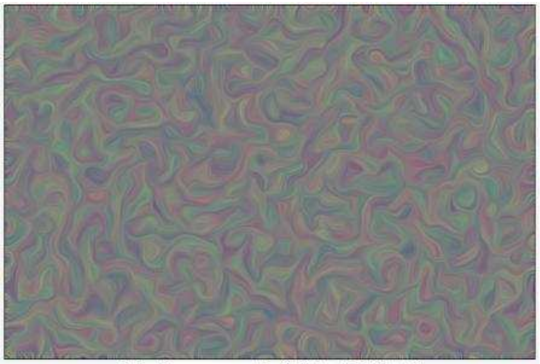
[0]: 'image.jpg' (640x427x1x3)

### • Example #2

```
image.jpg 100%,100%,1,2 rand[-1] -100,100 repeat 2 smooth[-1]
100,0.2,1,4,4 done warp[0] [-1],1,1,1
```



[0]: 'image.jpg' (640x427x1x3)



[1]: '[unnamed]' (640x427x1x2)

---

# snapshot3d

## Arguments:

- `_size>0,_zoom>=0,_backgroundR,_backgroundG,_backgroundB,_backgroundA,_fov_ang`

or
- `[background_image],zoom>=0,_fov_angle>=0`

## Description:

Take 2D snapshots of selected 3D objects.

Set `zoom` to 0 to disable object auto-scaling.

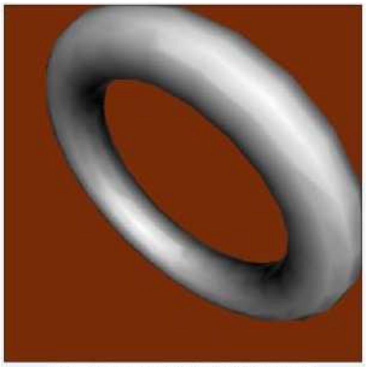
## Default values:

`size=1024`, `zoom=1`, `[background_image]=(default)` and `fov_angle=45`.

## Examples of use:

### • Example #1

```
torus3d 100,20 rotate3d 1,1,0,60 snapshot3d 400,1.2,128,64,32
```



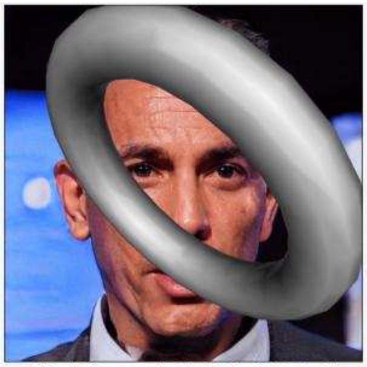
[0]: '[3D torus]' (400x400x1x3)

### • Example #2

```
torus3d 100,20 rotate3d 1,1,0,60 sample ? +snapshot3d[0] [1],1.2
```

[0]: '[3D torus]' (288 vert., 288 prim.)

[1]: 'portrait1' (800x800x1x3)



[2]: '[3D torus]_c1' (800x800x1x3)

---

# solarize

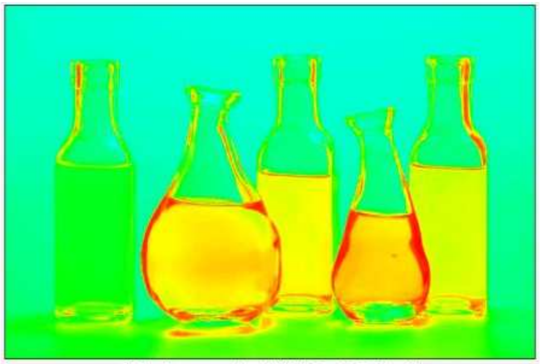
### No arguments

## Description:

Solarize selected images.

## Example of use:

```
image.jpg solarize
```



[0]: 'image.jpg' (640x427x1x3)

# solidify

## Arguments:

- `_smoothness[%]>=0,_diffusion_type={ 0:isotropic | 1:Delaunay-guided | 2:edge-oriented },_diffusion_iter>=0`

## Description:

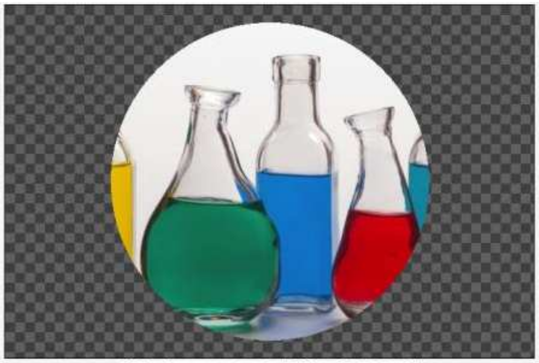
Solidify selected transparent images.

## Default values:

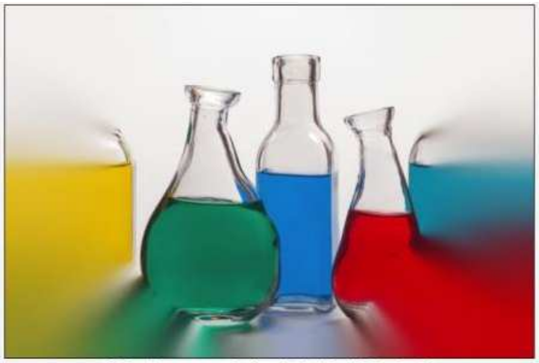
`smoothness=75%`, `diffusion_type=1` and `diffusion_iter=20`.

## Example of use:

```
image.jpg 100%,100% circle[-1] 50%,50%,25%,1,255 append c +solidify ,
display_rgba
```



[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# solve

Built-in command

## Arguments:

- `[image],_use_LU={ 0:SVD | 1:LU }`

## Description:

Solve linear system AX = B for selected B-matrices and specified A-matrix.

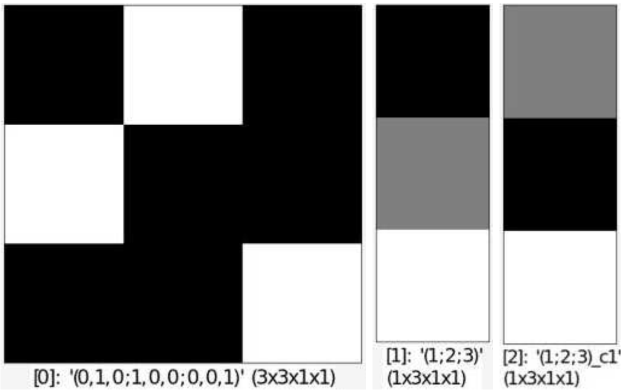
If the system is under- or over-determined, the least squares solution is returned.

## Default values:

`use_LU=0`.

## Example of use:

```
(0,1,0;1,0,0;0,0,1) (1;2;3) +solve[-1] [-2]
```


[0]: '(0,1,0;1,0,0;0,0,1)' (3x3x1x1)
[1]: '(1;2;3)' (1x3x1x1)
[2]: '(1;2;3)_c1' (1x3x1x1)

# solve_poisson

## Arguments:

- `"laplacian_command",_nb_iterations>=0,_time_step>0,_nb_scales>=0`

## Description:

Solve Poisson equation so that applying `laplacian[n]` is close to the result of `laplacian_command[n]`.

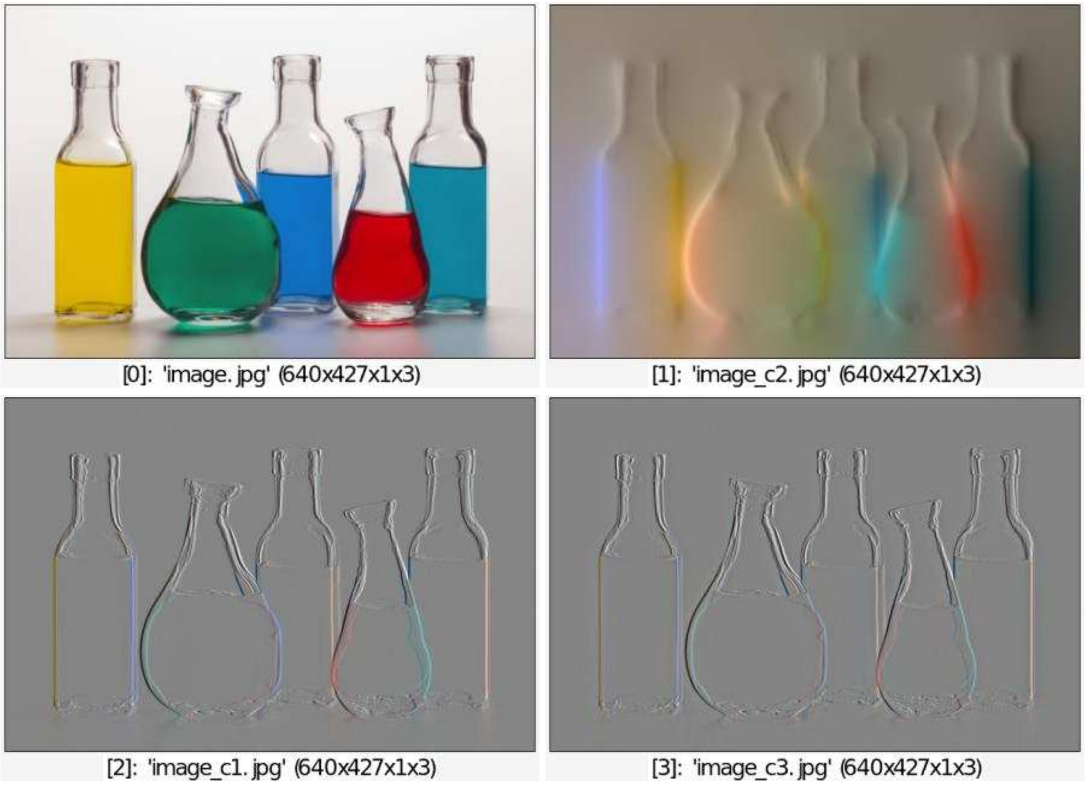
Solving is performed using a multi-scale gradient descent algorithm.
If `nb_scales=0`, the number of scales is automatically determined.

## Default values:

`nb_iterations=60`, `dt=5` and `nb_scales=0`.

## Example of use:

```
image.jpg command "foo : gradient x" +solve_poisson foo +foo[0]
+laplacian[1]
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c2.jpg' (640x427x1x3)
[2]: 'image_c1.jpg' (640x427x1x3)
[3]: 'image_c3.jpg' (640x427x1x3)

---

# sort

**Built-in command**

## Arguments:

- `_ordering={ + | - },_axis={ x | y | z | c }`

## Description:

Sort pixel values of selected images.

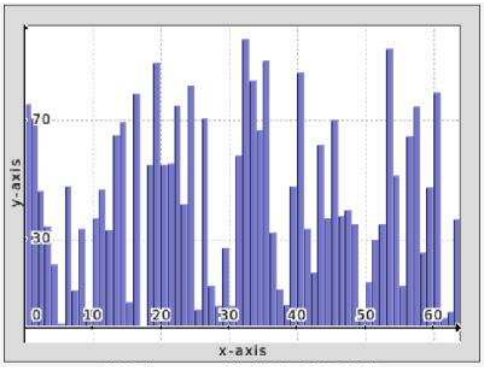
If `axis` is specified, the sorting is done according to the data of the first column/row/slice/channel of selected images.

## Default values:

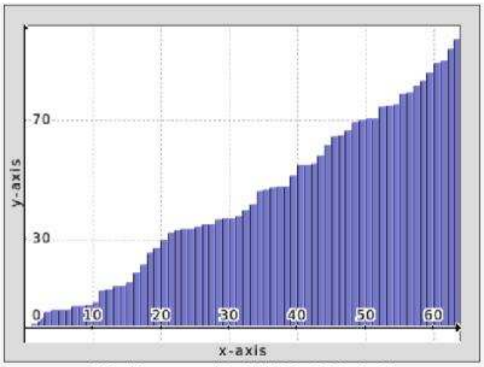
`ordering=+` and `axis=(undefined)`.

## Example of use:

```
64 rand 0,100 +sort display_graph 400,300,3
```



[0]: '[unnamed]' (400x300x1x3)



[1]: '[unnamed]_c1' (400x300x1x3)

# sort_list

## Arguments:

- `_ordering={ + | - },_criterion`

## Description:

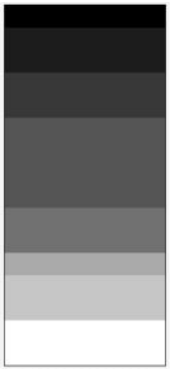
Sort list of selected images according to the specified image criterion.

## Default values:

`ordering=+`, `criterion=i`.

## Example of use:

```
(1;4;7;3;9;2;4;7;6;3;9;1;0;3;3;2) split y sort_list +,i append y
```



[0]: '(1;4;7;3;9;2;4;7;6;3;9;1;0;3;3;2)_c12'
(1x16x1x1)

# specl3d

## Arguments:

- `value>=0`

### Description:

Set lightness of 3D specular light.

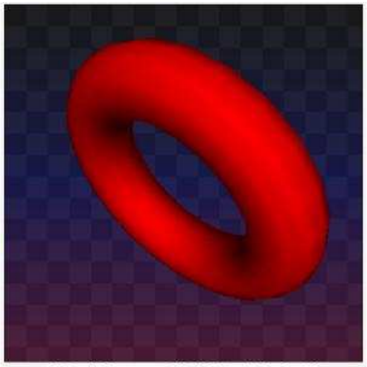
(*equivalent to shortcut command* `sl3d`).

### Default values:

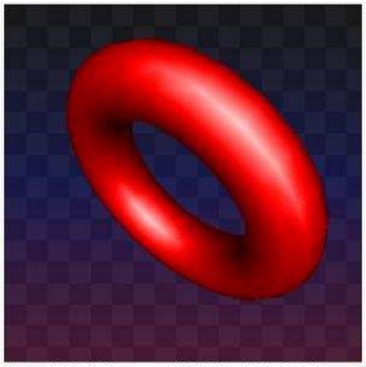
`value=0.15`.

### Example of use:

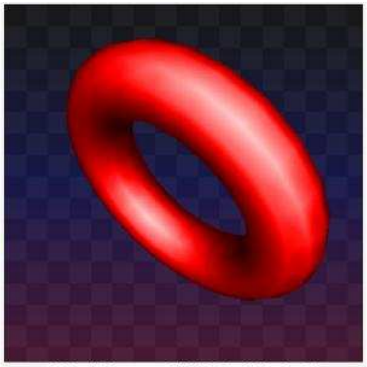
```
(0,0.3,0.6,0.9,1.2) repeat w { torus3d 100,30 rotate3d[-1] 1,1,0,60
color3d[-1] 255,0,0 specl3d {0,@$>} snapshot3d[-1] 400 } remove[0]
```



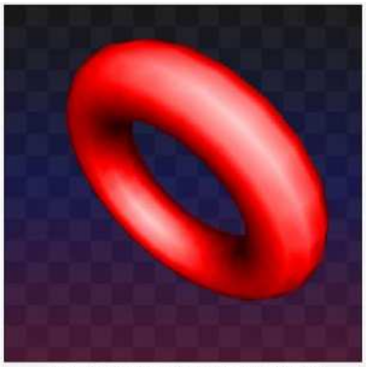
[0]: '[3D torus]' (400x400x1x3)



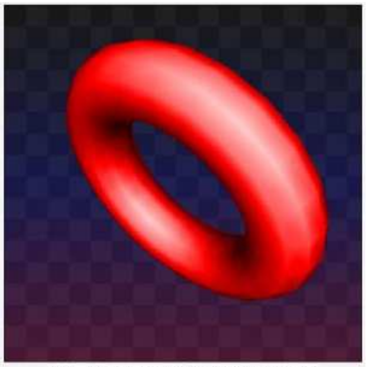
[1]: '[3D torus]' (400x400x1x3)



[2]: '[3D torus]' (400x400x1x3)



[3]: '[3D torus]' (400x400x1x3)



[4]: '[3D torus]' (400x400x1x3)

## specs3d

### Arguments:

- `value>=0`

### Description:

Set shininess of 3D specular light.

(*equivalent to shortcut command* `ss3d`).

### Default values:

`value=0.8`.

## Example of use:

```
(0,0.3,0.6,0.9,1.2) repeat w { torus3d 100,30 rotate3d[-1] 1,1,0,60
color3d[-1] 255,0,0 specs3d {0,@$>} snapshot3d[-1] 400 } remove[0]
```

[0]: '[3D torus]' (400x400x1x3)

[1]: '[3D torus]' (400x400x1x3)

[2]: '[3D torus]' (400x400x1x3)

[3]: '[3D torus]' (400x400x1x3)

[4]: '[3D torus]' (400x400x1x3)

---

# sphere3d

## Arguments:

- `radius,_nb_recursions!=0` or
- `radius,_nb_phi>=3,_nb_theta>=3`

## Description:

Input 3D sphere at (0,0,0), with specified geometry.

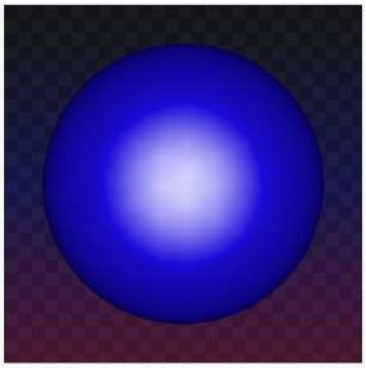
- If 2 arguments are specified:
  - If `nb_recursions>0`, the sphere is generated using recursive subdivisions of an **icosahedron**.
  - If `nb_recursions<0`, the sphere is generated using recursive subdividions of a **cube**.
- If 3 arguments are specified, the sphere is generated using spherical coordinates discretization.

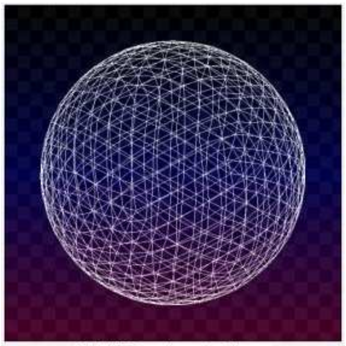
## Default values:

`nb_recursions=3`.

## Example of use:

```
sphere3d 100 +primitives3d 1 color3d[-2] ${-rgb}
```


[0]: '[3D sphere]' (642 vert., 1280 prim.)


[1]: '[3D sphere]_c1'
(642 vert., 1920 prim.)

# spherical3d

## Arguments:

- `"radius_function(phi,theta)",_nb_recursions!=0` or
- `"radius_function(phi,theta)",_nb_phi>=3,_nb_theta>=3`

## Description:

Input 3D spherical object at (0,0,0), with specified geometry.

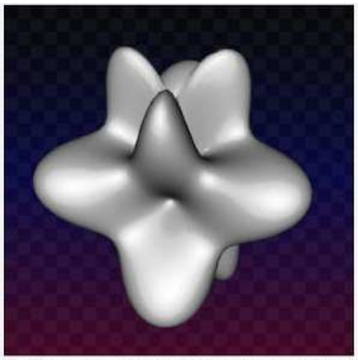
Second and third arguments are the same as in command **sphere3d**.

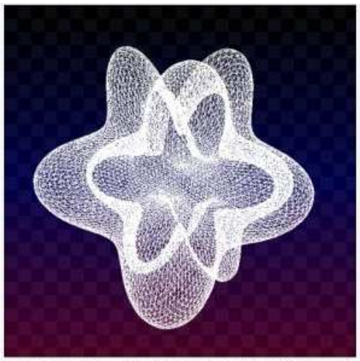
## Default values:

`nb_recursions=5`.

## Example of use:

```
spherical3d "abs(1+0.5*cos(3*phi)*sin(4*theta))" +primitives3d 1
```


[0]: '[3D spherical surface 'abs(1+0.5*cos(3*phi)*sin(4*theta))']'
(10242 vert., 20480 prim.)


[1]: '[3D spherical surface 'abs(1+0_c1.5*cos(3*phi)*sin(4*theta))']'
(10242 vert., 30720 prim.)

# spherize

## Arguments:

- `_radius[%]>=0,_strength,_smoothness[%]>=0,_center_x[%],_center_y[%],_ratio_x/y>0,_angle,_interpolation`

## Description:

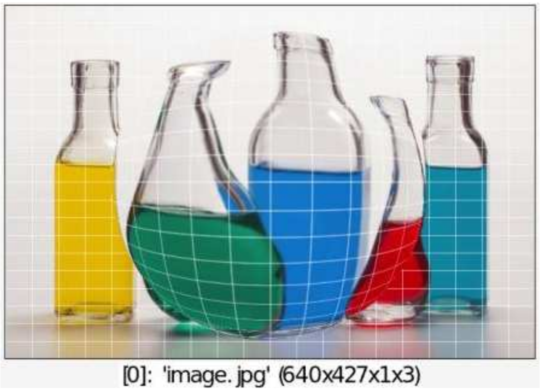
Apply spherize effect on selected images.

## Default values:

`radius=50%`, `strength=1`, `smoothness=0`, `center_x=center_y=50%`, `ratio_x/y=1`, `angle=0` and `interpolation=1`.

## Example of use:

```
image.jpg grid 5%,5%,0,0,0.6,255 spherize ,
```



[0]: 'image.jpg' (640x427x1x3)

# spiralbw

## Arguments:

- `width>0,_height>0,_is_2dcoords={ 0 | 1 }`

## Description:

Input a 2D rectangular spiral image with specified size.

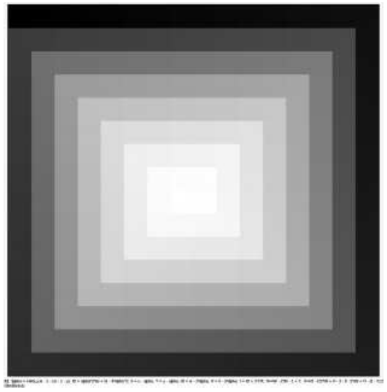
## Default values:

`height=width` and `is_2dcoords=0`.

## Examples of use:

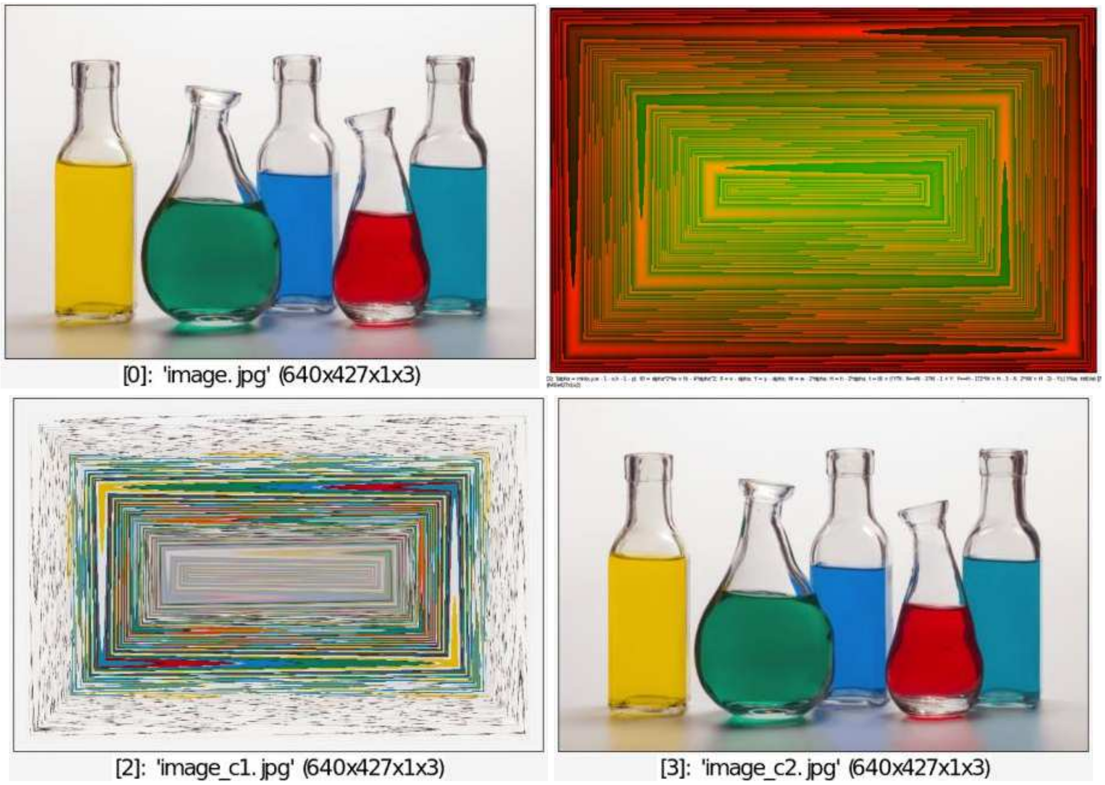
- **Example #1**

```
spiralbw 16
```



- **Example #2**

```
image.jpg spiralbw {[w,h]},1 +warp[0] [1],0,1,1 +warp[2] [1],2,1,1
```



[0]: 'image.jpg' (640x427x1x3)

[2]: 'image_c1.jpg' (640x427x1x3)

[3]: 'image_c2.jpg' (640x427x1x3)

---

# spline

## Arguments:

- `x0[%],y0[%],u0[%],v0[%],x1[%],y1[%],u1[%],v1[%],_opacity,_color1,...`

## Description:

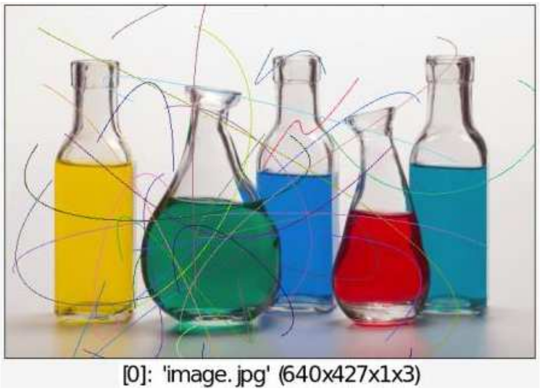
Draw specified colored spline curve on selected images (cubic hermite spline).

## Default values:

`opacity=1` and `color1=0`.

## Example of use:

```
image.jpg repeat 30 { spline {u(100)}%,{u(100)}%,{u(-600,600)},
{u(-600,600)},{u(100)}%,{u(100)}%,{u(-600,600)},{u(-600,600)},1,${-
RGB} }
```



[0]: 'image.jpg' (640x427x1x3)

---

# spline3d

## Arguments:

- `x0[%],y0[%],z0[%],u0[%],v0[%],w0[%],x1[%],y1[%],z1[%],u1[%],v1[%],w1[%],_nb_v`

## Description:

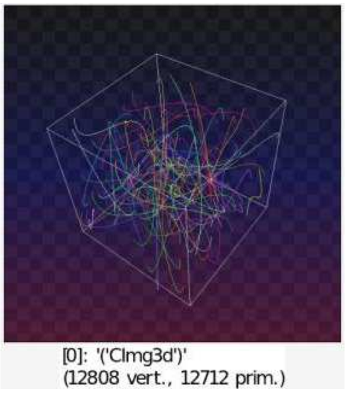
Input 3D spline with specified geometry.

## Default values:

`nb_vertices=128`.

## Example of use:

```
repeat 100 { spline3d {u},{u},{u},{u},{u},{u},{u},{u},{u},{u},{u},
{u},128 color3d[-1] ${-rgb} } box3d 1 primitives3d[-1] 1 add3d
```



[0]: '('CImg3d')'
(12808 vert., 12712 prim.)

# split

**Built-in command**

## Arguments:

- `{ x | y | z | c }...{ x | y | z | c },_split_mode` or
- `keep_splitting_values={ + | - },_{ x | y | z | c }...{ x | y | z | c },value1,_value2,...` or
- `(no arg)`

## Description:

Split selected images along specified axes, or regarding to a sequence of scalar values

(optionally along specified axes too).

(*equivalent to shortcut command* `s`).

`split_mode` can be ***{ 0:split according to constant values | >0:split in N parts | <0:split in parts of size -N }***.

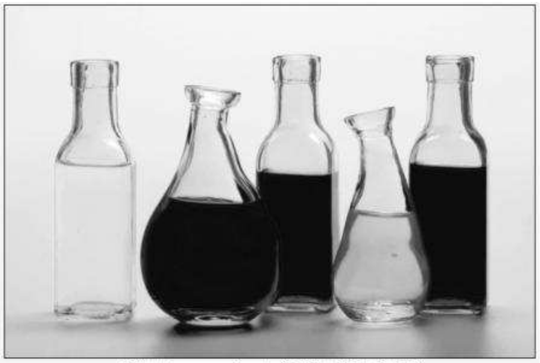
## Default values:

`split_mode=-1`.

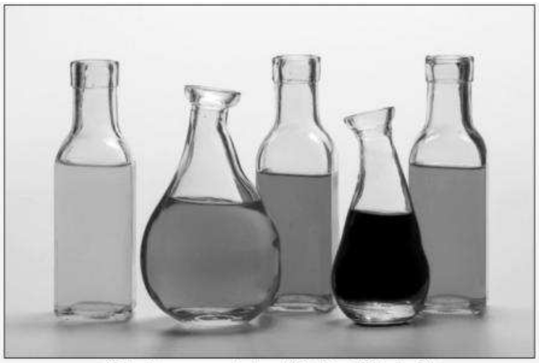
## Examples of use:

### • Example #1

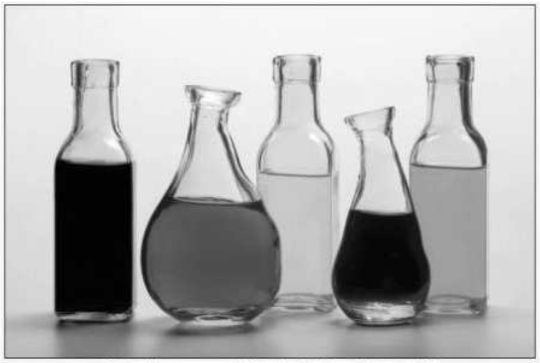
```
image.jpg split c
```



[0]: 'image.jpg' (640x427x1x1)



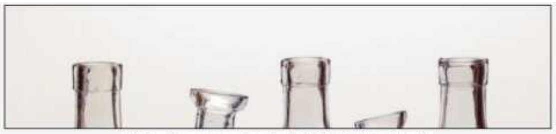
[1]: 'image_c1.jpg' (640x427x1x1)



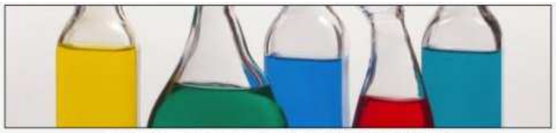
[2]: 'image_c2.jpg' (640x427x1x1)

- **Example #2**

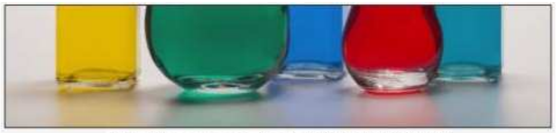
```
image.jpg split y,3
```



[0]: 'image.jpg' (640x143x1x3)



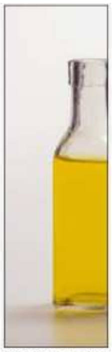
[1]: 'image_c1.jpg' (640x142x1x3)



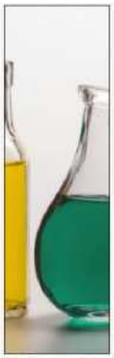
[2]: 'image_c2.jpg' (640x142x1x3)

- **Example #3**

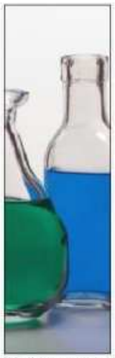
```
image.jpg split x,-128
```



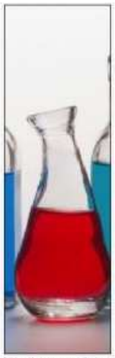
[0]: 'image.jpg' (128x427x1x3)



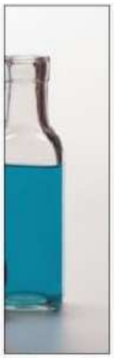
[1]: 'image_c1.jpg' (128x427x1x3)



[2]: 'image_c2.jpg' (128x427x1x3)



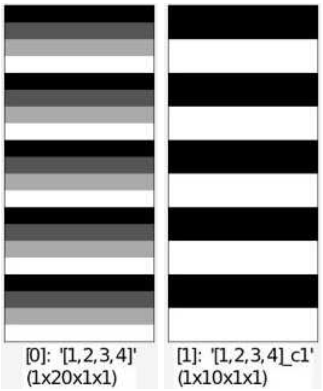
[3]: 'image_c3.jpg' (128x427x1x3)



[4]: 'image_c4.jpg' (128x427x1x3)

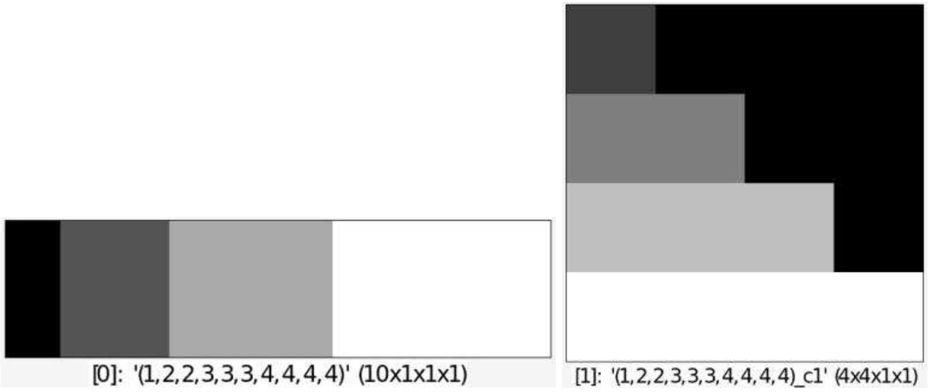
- **Example #4**

```
1,20,1,1,"1,2,3,4" +split -,2,3 append[1--1] y
```



- **Example #5**

```
(1,2,2,3,3,3,4,4,4,4) +split x,0 append[1--1] y
```


[0]: '(1,2,2,3,3,3,4,4,4,4)' (10x1x1x1)
[1]: '(1,2,2,3,3,3,4,4,4,4)_c1' (4x4x1x1)

# split3d

### No arguments

## Description:

Split selected 3D objects into feature vectors :

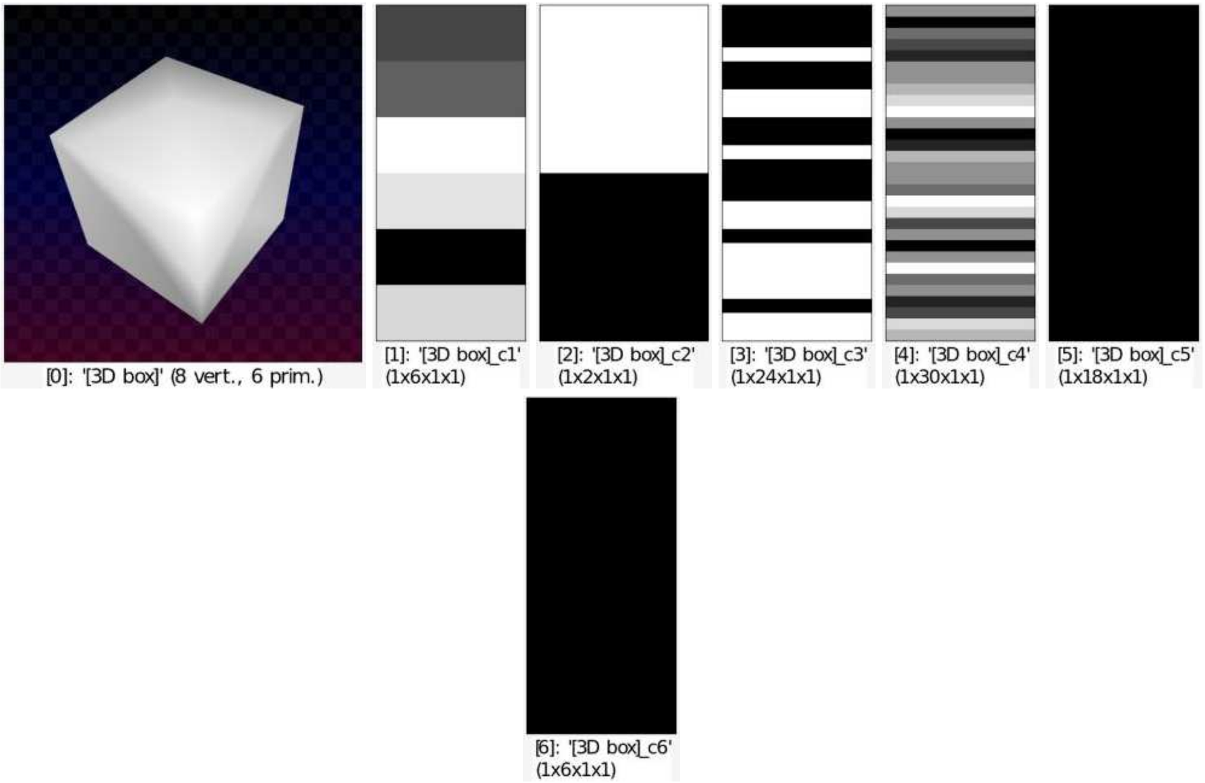
***`{ header, sizes, vertices, primitives, colors, opacities }`***.

(*equivalent to shortcut command* `s3d`).

To recreate the 3D object, append all produced images along the y-axis (with command `append y`).

## Example of use:

```
box3d 100 +split3d
```


[0]: '[3D box]' (8 vert., 6 prim.)
[1]: '[3D box]_c1' (1x6x1x1)
[2]: '[3D box]_c2' (1x2x1x1)
[3]: '[3D box]_c3' (1x24x1x1)
[4]: '[3D box]_c4' (1x30x1x1)
[5]: '[3D box]_c5' (1x18x1x1)
[6]: '[3D box]_c6' (1x6x1x1)

# split_alpha

## Arguments:

- `_nb_scales[%]={ 0:auto | -S<0 | N>0 },_subsample={ 0:no | 1:yes }, 0<=_anisotropy<=1,0<=_minimize_alpha<=1`

## Description:

Split selected images into alpha detail scales.

If `nb_scales==-S`, the lowest scale has a size of at least `SxS`.
Parameter `anisotropy` is only considered when `subsample=0`.
Image reconstruction is done with command **merge_alpha**.

## Default values:

`nb_scales=0`, `subsample=0`, `anisotropy=0` and `minimize_alpha=1`.

# split_colors

## Arguments:

- `_tolerance>=0,_max_nb_outputs>0,_min_area>0`

## Description:

Split selected images as several image containing a single color.

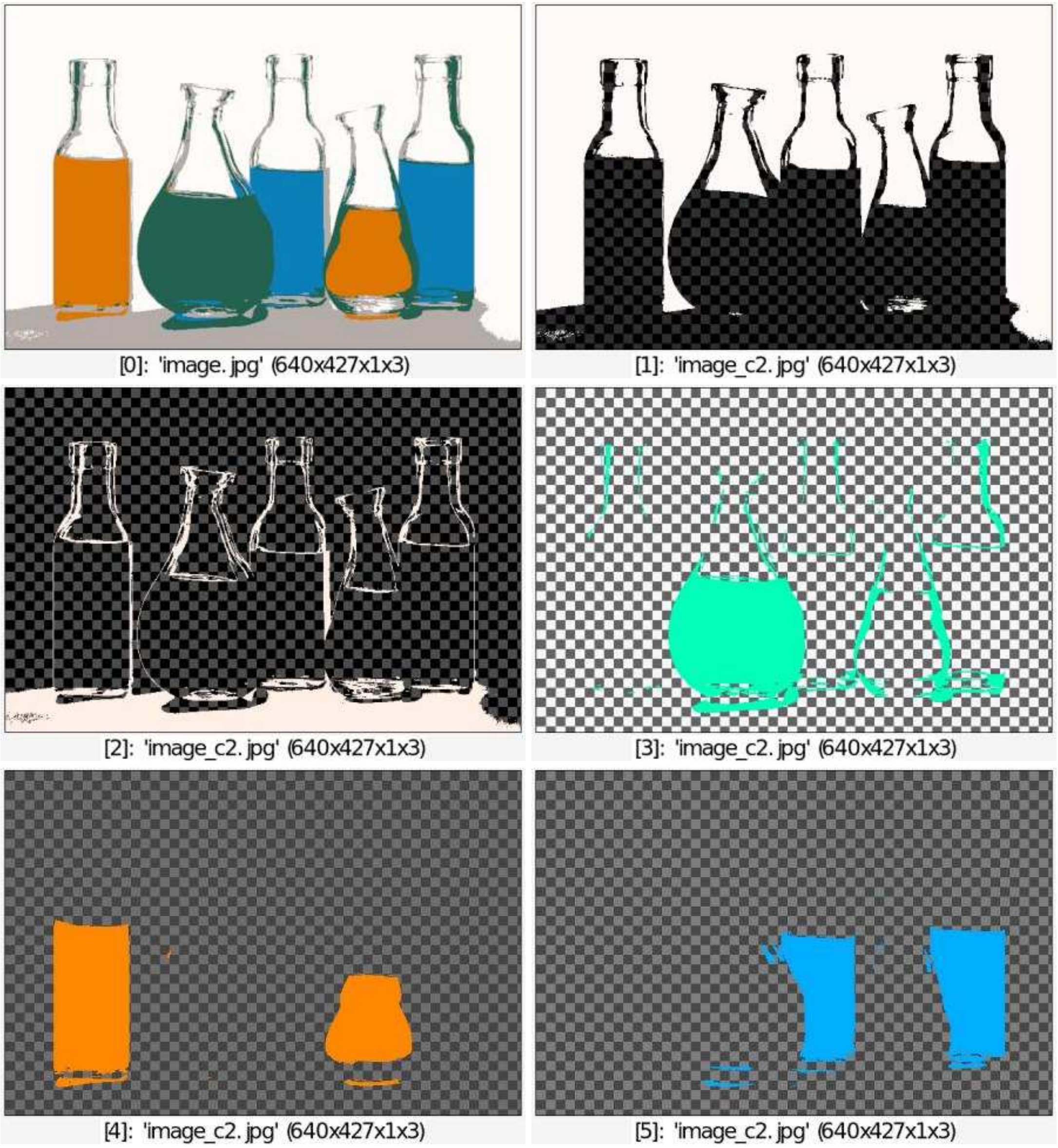
One selected image can be split as at most `max_nb_outputs` images.
Output images are sorted by decreasing area of extracted color regions and have an additional alpha-channel.

## Default values:

`tolerance=0`, `max_nb_outputs=256` and `min_area=8`.

## Example of use:

```
image.jpg quantize 5 +split_colors , display_rgba
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c2.jpg' (640x427x1x3)

[2]: 'image_c2.jpg' (640x427x1x3)

[3]: 'image_c2.jpg' (640x427x1x3)

[4]: 'image_c2.jpg' (640x427x1x3)

[5]: 'image_c2.jpg' (640x427x1x3)

# split_details

## Arguments:

- `_nb_scales[%]={ 0:auto | -S<0 | N>0 },_base_scale[%]>=0,_detail_scale[%]>=0`

## Description:

Split selected images into `nb_scales` detail scales.

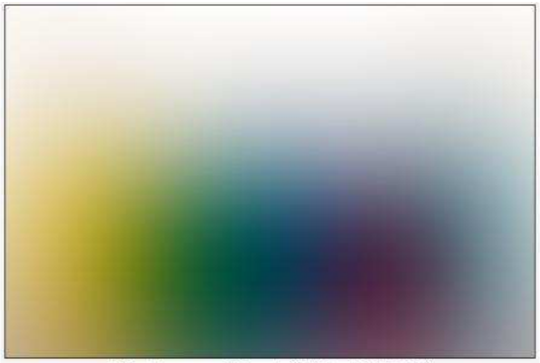
If `base_scale` = `detail_scale` = 0, the image decomposition is done with 'a trous' wavelets. Otherwise, it uses laplacian pyramids with linear standard deviations.

## Default values:

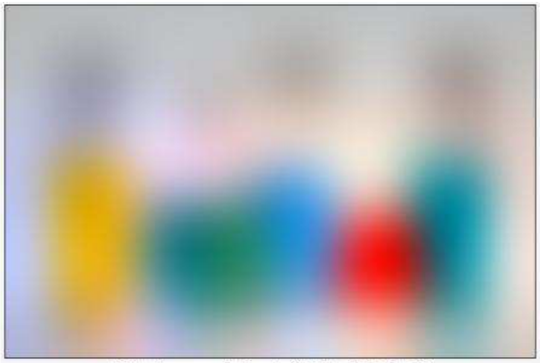
`nb_scales=0`, `base_scale=0` and `detail_scale=0`.

## Example of use:

```
image.jpg split_details ,
```



[0]: 'image_0.jpg' (640x427x1x3)



[1]: 'image_1.jpg' (640x427x1x3)

[2]: 'image_2.jpg' (640x427x1x3)

[3]: 'image_3.jpg' (640x427x1x3)

[4]: 'image_4.jpg' (640x427x1x3)

[5]: 'image_5.jpg' (640x427x1x3)

[6]: 'image_6.jpg' (640x427x1x3)

[7]: 'image_7.jpg' (640x427x1x3)

# split_freq

## Arguments:

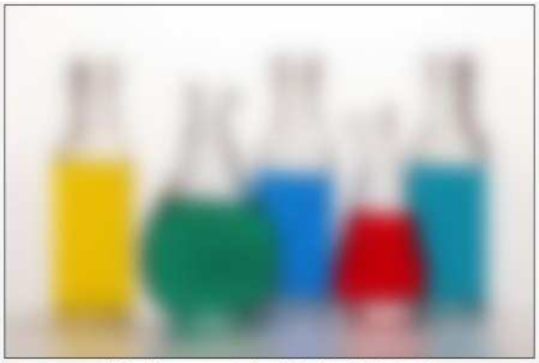
- `smoothness>0[%]`

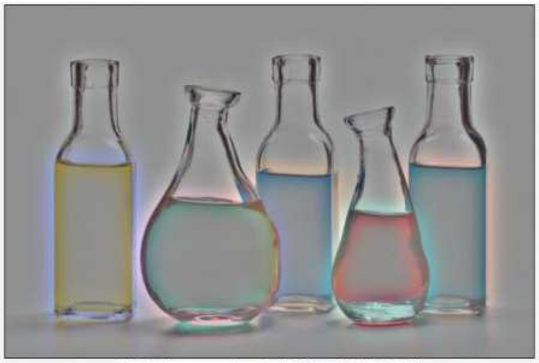
## Description:

Split selected images into low and high frequency parts.

## Example of use:

```
image.jpg split_freq 2%
```


[0]: 'image_c1.jpg' (640x427x1x3)


[1]: 'image.jpg' (640x427x1x3)

# split_opacity

**No arguments**

## Description:

Split color and opacity parts of selected images.

This command returns 1 or 2 images for each selected image, whether it has an opacity channel or not.

---

# split_tiles

## Arguments:

- `M!=0,_N!=0,_is_homogeneous={ 0 | 1 }`

## Description:

Split selected images as a MxN array of tiles.

If M or N is negative, it stands for the tile size instead.

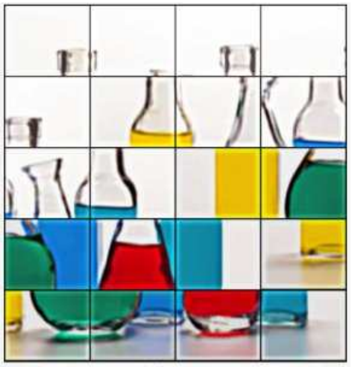
## Default values:

`N=M` and `is_homogeneous=0`.

## Example of use:

```
image.jpg +local split_tiles 5,4 blur 3,0 sharpen 700 append_tiles
4,5 done
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (512x535x1x3)

---

# split_vector

## Arguments:

- `keep_splitting_values={ + | - },value1,_value2,...`

### Description:

Split selected images into multiple parts, where specified vector `[value1,_value2,...]` is the separator.

## sponge

### Arguments:

- `_size>0`

### Description:

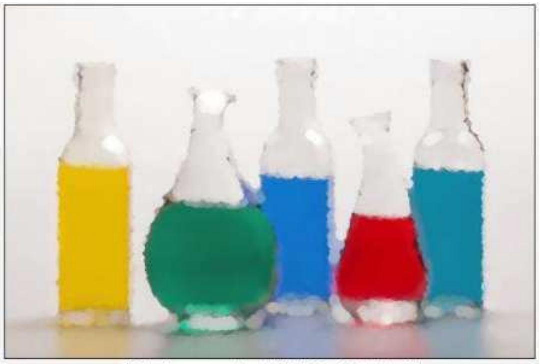
Apply sponge effect on selected images.

### Default values:

`size=13`.

### Example of use:

```
image.jpg sponge ,
```



[0]: 'image.jpg' (640x427x1x3)

## spread

### Arguments:

- `_dx>=0,_dy>=0,_dz>=0`

### Description:

Spread pixel values of selected images randomly along x,y and z.

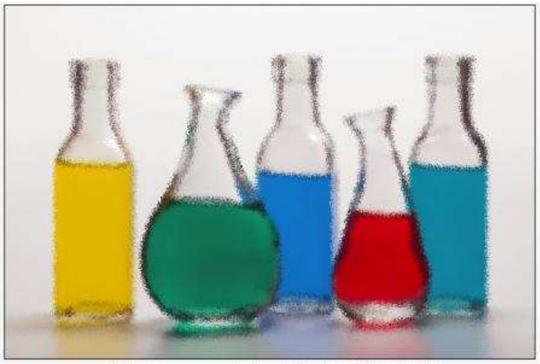
### Default values:

`dx=3`, `dy=dx` and `dz=0`.

### Example of use:

```
image.jpg +spread 3
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

## sprite3d

**No arguments**

### Description:

Convert selected images as 3D sprites.

Selected images with alpha channels are managed.

### Example of use:

```
image.jpg sprite3d
```

[0]: 'image.jpg' (1 vert., 1 prim.)

---

## sprites3d

### Arguments:

- `[sprite],_sprite_has_alpha_channel={ 0 | 1 }`

## Description:

Convert selected 3D objects as a sprite cloud.

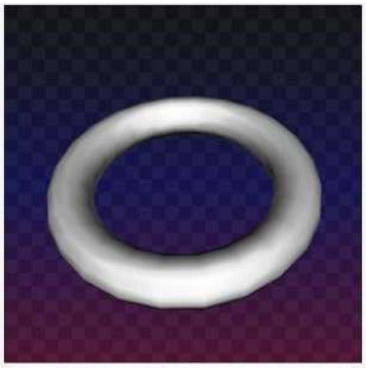
Set `sprite_has_alpha_channel` to 1 to make the last channel of the selected sprite be a transparency mask.

## Default values:

`mask_has_alpha_channel=0`.

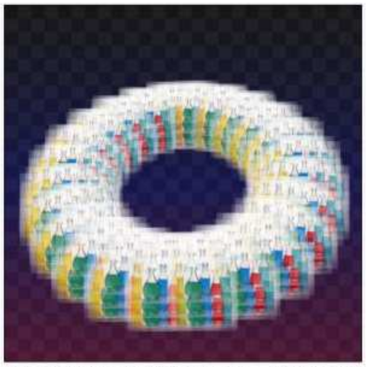
## Example of use:

```
torus3d 100,20 image.jpg rescale2d[-1] ,64 100%,100% gaussian[-1]
30%,30% *[-1] 255 append[-2,-1] c +sprites3d[0] [1],1
display_rgba[-2]
```



[0]: '[3D torus]' (288 vert., 288 prim.)

[1]: 'image.jpg' (96x64x1x3)



[2]: '[3D torus]_c1' (288 vert., 288 prim.)

# sqr

Built-in command

### No arguments

## Description:

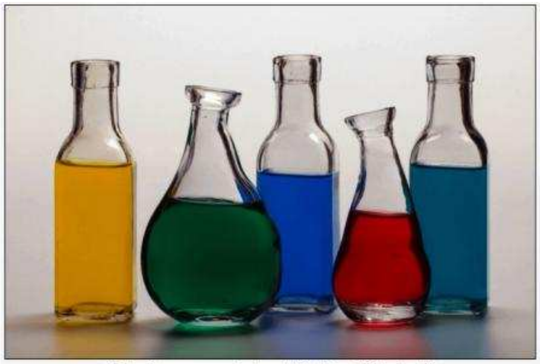
Compute the pointwise square function of selected images.

### Examples of use:

- **Example #1**

```
image.jpg +sqr
```

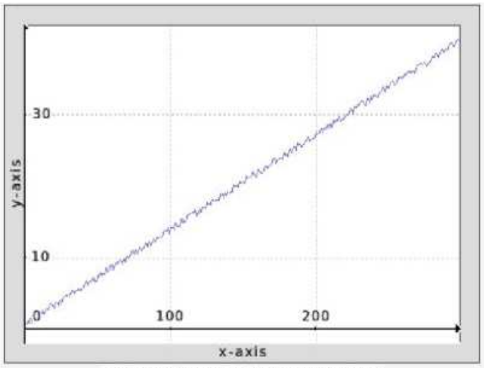
[0]: 'image.jpg' (640x427x1x3)



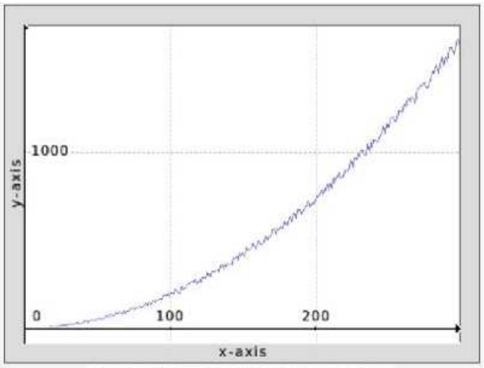
[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
300,1,1,1,'40*x/w+u' +sqr display_graph 400,300
```



[0]: '[40*x/w+u]' (400x300x1x3)



[1]: '[40*x/w+u]_c1' (400x300x1x3)

---

## sqrt

**Built-in command**

**No arguments**

### Description:

Compute the pointwise square root of selected images.

### Examples of use:

- **Example #1**

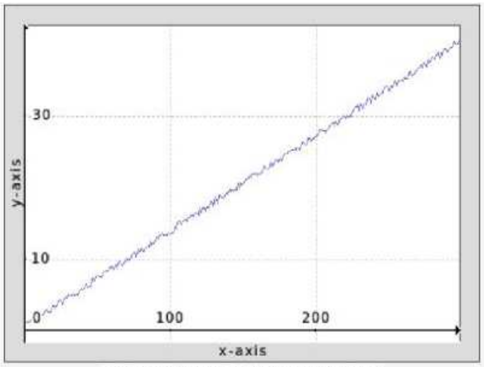
```
image.jpg +sqrt
```

[0]: 'image.jpg' (640x427x1x3)

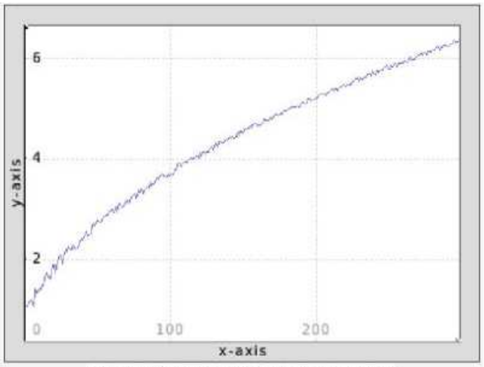
[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #2**

```
300,1,1,1,'40*x/w+u' +sqrt display_graph 400,300
```



[0]: '[40*x/w+u]' (400x300x1x3)



[1]: '[40*x/w+u]_c1' (400x300x1x3)

# srand

Built-in command

## Arguments:

- `value` or
- `(no arg)`

## Description:

Set random generator seed.

If no argument is specified, a random value is used as the random generator seed.

# srgb2lab

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from sRGB to Lab.

## Default values:

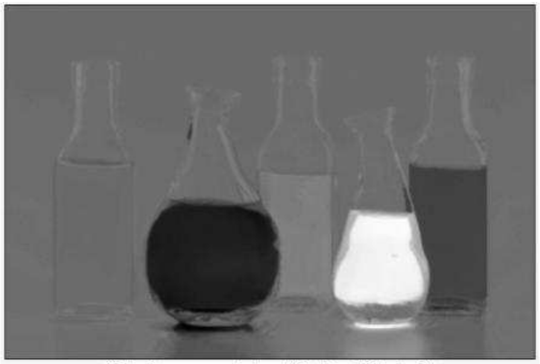
`illuminant=2`.

## Examples of use:

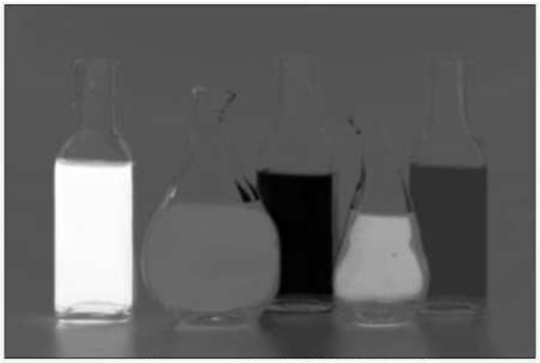
### • Example #1

```
image.jpg srgb2lab split c
```

[0]: 'image. jpg' (640x427x1x1)



[1]: 'image_c1. jpg' (640x427x1x1)

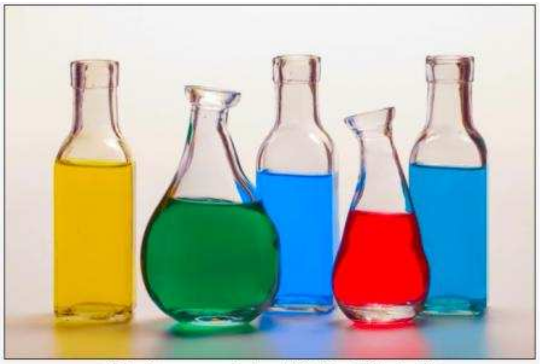


[2]: 'image_c2. jpg' (640x427x1x1)

### • Example #2

```
image.jpg srgb2lab +split c mul[-2,-1] 2.5 append[-3--1] c lab2srgb
```

[0]: 'image. jpg' (640x427x1x3)



[1]: 'image_c1. jpg' (640x427x1x3)

# srgb2lab8

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from sRGB to Lab8.

## Default values:

`illuminant=2`.

# srgb2rgb

**No arguments**

## Description:

Convert color representation of selected images from sRGB to linear RGB.

# ssd_patch

## Arguments:

- `[patch],_use_fourier={ 0 | 1 },_boundary_conditions`

## Description:

Compute fields of SSD between selected images and specified patch.

Argument `boundary_conditions` is valid only when `use_fourier=0`.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

## Default values:

`use_fourier=0` and `boundary_conditions=0`.

## Example of use:

```
image.jpg +crop 20%,20%,35%,35% +ssd_patch[0] [1],0,0
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (97x65x1x3)

[2]: 'image_c1.jpg' (640x427x1x1)

---

# ssim

## Arguments:

- `[reference],_patch_size>0,_max_value>0`

## Description:

Compute the Structural Similarity Index Measure (SSIM) between selected images and specified reference image.

This command does not modify the images, it just returns a value or a list of values in the status. When `downsampling_factor` is specified with a ending `%`, its value is equal to `1+(patch_size-1)*spatial_factor%`.

SSIM is a measure introduced int the following paper:
*Wang, Zhou, et al.*, "Image quality assessment: from error visibility to structural similarity.",
in IEEE transactions on image processing 13.4 (2004): 600-612.

The implementation of this command is a direct translation of the reference code (in Matlab), found at :
https://ece.uwaterloo.ca/~z70wang/research/ssim/

## Default values:

`patch_size=11`, and `max_value=255`.

# ssim_matrix

## Arguments:

- `_patch_size>0,_max_value>0`

## Description:

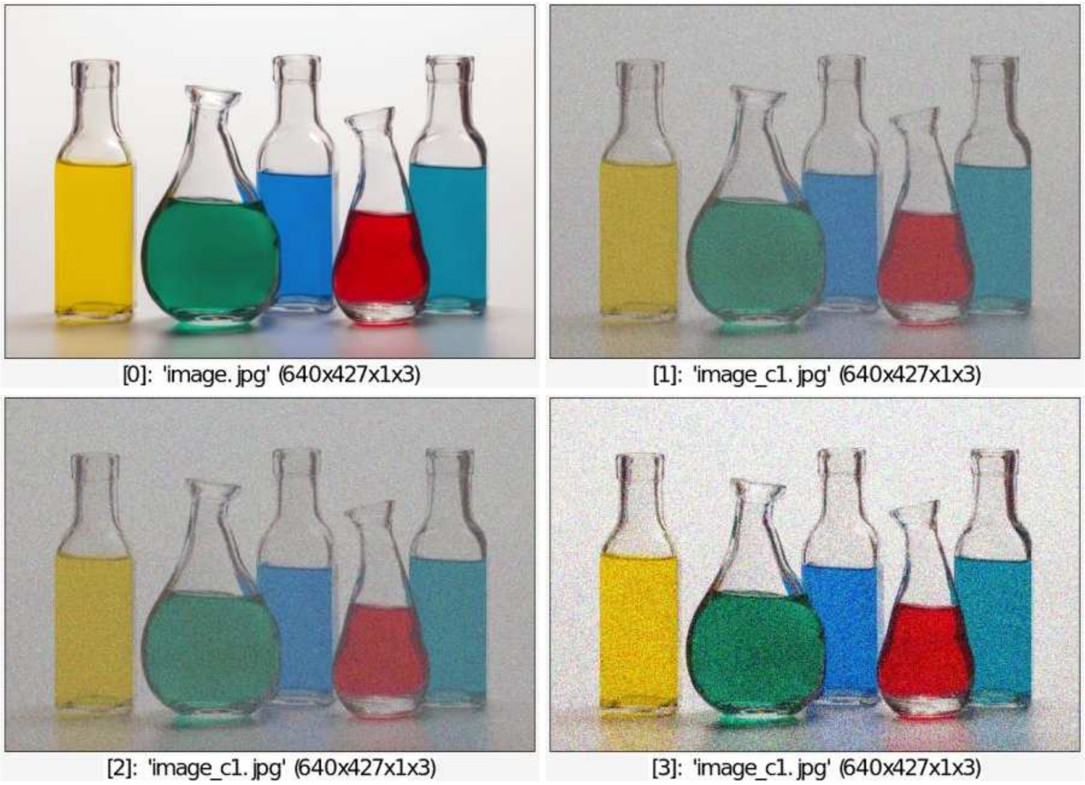
Compute SSIM (Structural Similarity Index Measure) matrix between selected images.

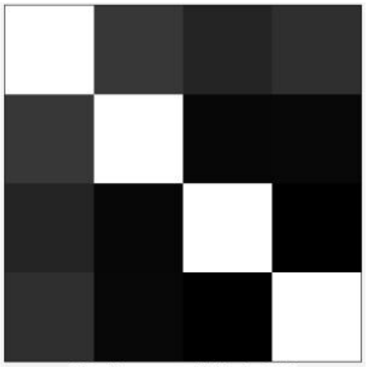
## Default values:

`patch_size=11`, and `max_value=255`.

## Example of use:

```
image.jpg +noise 30 +noise[0] 35 +noise[0] 38 cut. 0,255 +ssim_matrix
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)
[2]: 'image_c1.jpg' (640x427x1x3)
[3]: 'image_c1.jpg' (640x427x1x3)



[4]: '[unnamed]' (4x4x1x1)

---

# stained_glass

## Arguments:

- `_edges[%]>=0, shading>=0, is_thin_separators={ 0 | 1 }`

## Description:

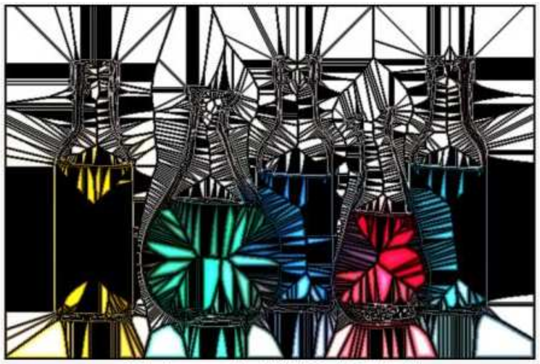
Generate stained glass from selected images.

## Default values:

`edges=40%`, `shading=0.2` and `is_precise=0`.

## Example of use:

```
image.jpg stained_glass 20%,1 cut 0,20
```



[0]: 'image.jpg' (640x427x1x3)

---

# star3d

## Arguments:

- `_nb_branches>0,0<=_thickness<=1`

### Description:

Input 3D star at position `(0,0,0)`, with specified geometry.

### Default values:

`nb_branches=5` and `thickness=0.38`.

### Example of use:

```
star3d , +primitives3d 1 color3d[-2] ${-rgb}
```

[0]: '[3D star]' (11 vert., 10 prim.)

[1]: '[3D star]_c1' (11 vert., 20 prim.)

## stars

### Arguments:

- `_density[%]>=0,_depth>=0,_size>0,_nb_branches>=1,0<=_thickness<=1,_smoothness`

### Description:

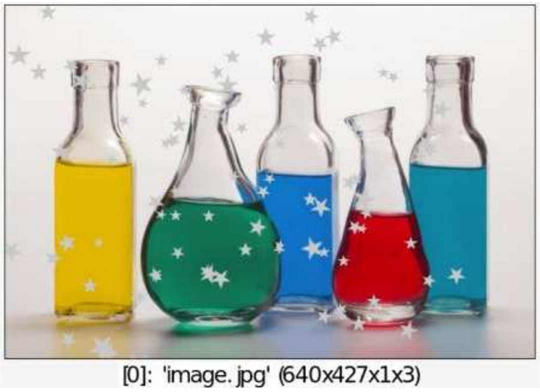
Add random stars to selected images.

### Default values:

`density=10%`, `depth=1`, `size=32`, `nb_branches=5`, `thickness=0.38`, `smoothness=0.5`, `R=G=B=200` and `opacity=1`.

### Example of use:

```
image.jpg stars ,
```



[0]: 'image.jpg' (640x427x1x3)

# status

Built-in command

## Arguments:

- `status_string`

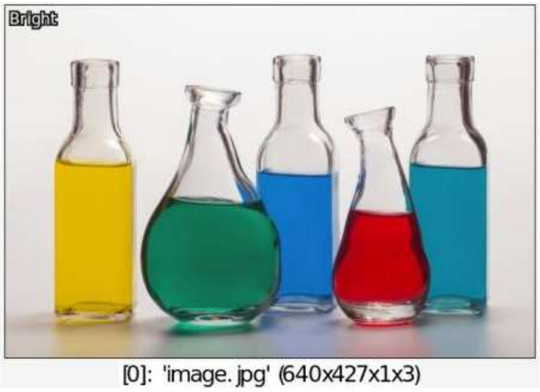
## Description:

Set the current status. Used to define a returning value from a function.

(*equivalent to shortcut command* `u`).

## Example of use:

```
image.jpg command "foo : u0=Dark u1=Bright status ${u{ia>=128}}"
text_outline ${-foo},2,2,23,2,1,255
```



[0]: 'image.jpg' (640x427x1x3)

# std_noise

**No arguments**

## Description:

Return the estimated noise standard deviation of the last selected image.

# stencil

## Arguments:

- `_radius[%]>=0,_smoothness>=0,_iterations>=0`

## Description:

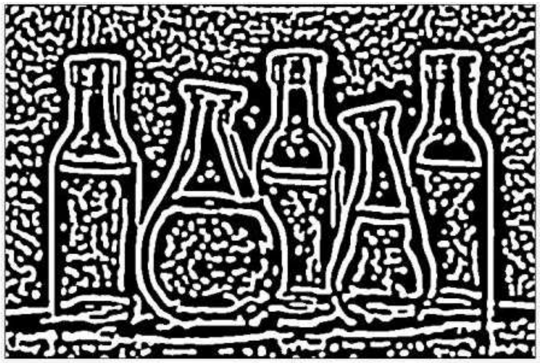
Apply stencil filter on selected images.

## Default values:

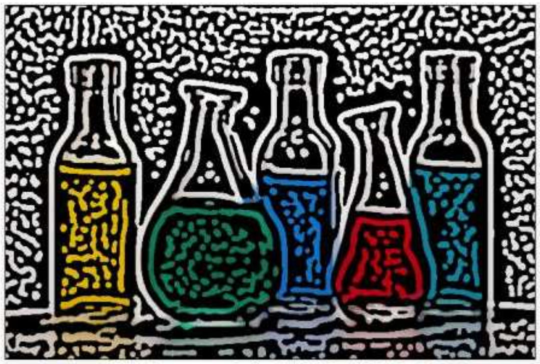
`radius=3`, `smoothness=1` and `iterations=8`.

## Example of use:

```
image.jpg +norm stencil. 2,1,4 +mul rm[0]
```



[0]: 'image_c1.jpg' (640x427x1x1)



[1]: 'image_c1.jpg' (640x427x1x3)

# stencilbw

## Arguments:

- `_edges>=0,_smoothness>=0`

## Description:

Apply B&W stencil effect on selected images.

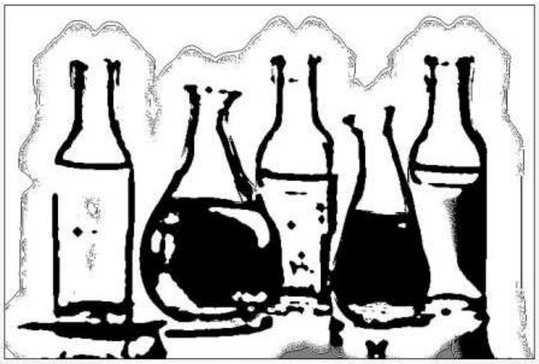
## Default values:

`edges=15` and `smoothness=10`.

## Example of use:

```
image.jpg +stencilbw 40,4
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x1)

## store

Built-in command

### Arguments:

- `_is_compressed={ 0 | 1 },variable_name1,_variable_name2,...`

### Description:

Store selected images into one or several named variables.

Selected images are transferred to the variables, and are so removed from the image list.
(except if the prepended variant of the command `+store[selection]` is used).
If a single variable name is specified, all images of the selection are assigned
to the named variable. Otherwise, there must be as many variable names as images
in the selection, and each selected image is assigned to each specified named variable.
Use command `input $variable_name` to bring the stored images back in the list.

### Default values:

`is_compressed=0`.

This command has a **tutorial page**.

### Example of use:

```
sample eagle,earth store img1,img2 input $img2 $img1
```

[0]: 'earth' (500x500x1x3)

[1]: 'eagle' (520x480x1x3)

---

# str

## Arguments:

- `string`

## Description:

Print specified string into its binary, octal, decimal and hexadecimal representations.

---

# str2hex

## Arguments:

- `"string"`

## Description:

Convert specified string argument into a sequence of hexadecimal values (returned as a string).

## See also:

**hex2str**.

## Example of use:

```
hex=${"str2hex \"Hello my friends\""} echo $hex
```

```
[gmic]-0./ Start G'MIC interpreter.
[gmic]-0./ 48656c6c6f206d7920667269656e6473
[gmic]-0./ End G'MIC interpreter.
```

---

# strbuffer

## Arguments:

- `buffer_size`

## Description:

Return a string describing a size for the specified buffer size.

---

# strcapitalize

## Arguments:

- `string`

## Description:

Capitalize specified string.

---

# strcasevar

## Arguments:

- `"string"`

## Description:

Return a simplified version of the specified string, that can be used as a variable name.

(version that keeps original case of specified string, no longer than 128 chars).

---

# strclut

## Arguments:

- `"string"`

## Description:

Return simplified version of the specified string that can be used as a CLUT name.

---

# strcontains

## Arguments:

- `string1,string2`

## Description:

Return 1 if the first string contains the second one.

---

# streamline3d

Built-in command

## Arguments:

- `x[%],y[%],z[%],_L>=0,_dl>0,_interpolation,_is_backward={ 0 | 1 },_is_oriented={ 0 | 1 }` or
- `'formula',x,y,z,_L>=0,_dl>0,_interpolation,_is_backward={ 0 | 1 },_is_oriented={ 0 | 1 }`

## Description:

Extract 3D streamlines from selected vector fields or from specified formula.

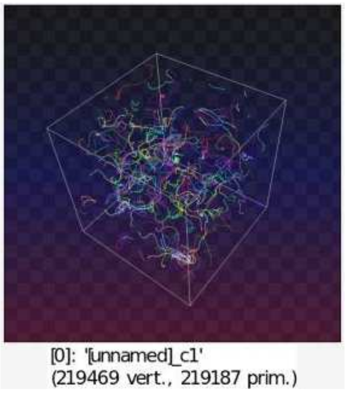
`interpolation` can be ***{ 0:nearest integer | 1:1st-order | 2:2nd-order | 3:4th-order }***.

## Default values:

`dl=0.1`, `interpolation=2`, `is_backward=0` and `is_oriented=0`.

## Example of use:

```
100,100,100,3 rand -10,10 blur 3 repeat 300 { +streamline3d[0] {u(100)},{u(100)},{u(100)},1000,1,1 color3d[-1] ${-rgb} } remove[0] box3d 100 primitives3d[-1] 1 add3d
```



[0]: '[unnamed]_c1'
(219469 vert., 219187 prim.)

---

# stripes_y

## Arguments:

- `_frequency>=0`

## Description:

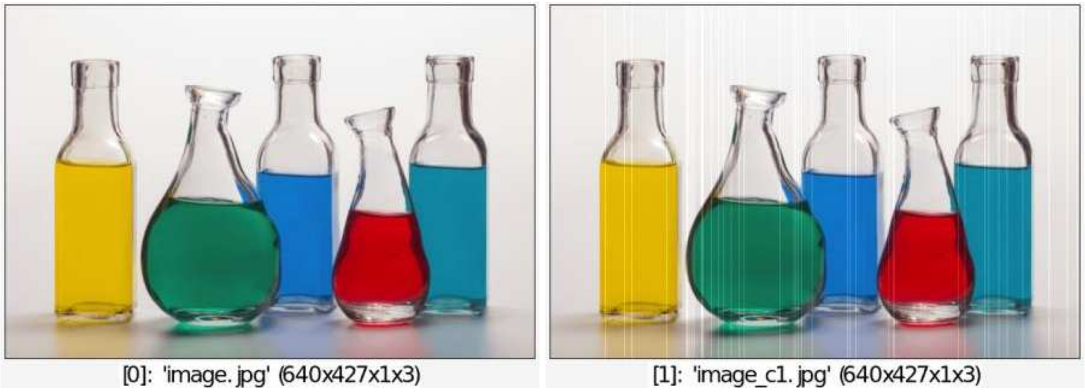
Add vertical stripes to selected images.

## Default values:

`frequency=10`.

## Example of use:

```
image.jpg +stripes_y ,
```


[0]: 'image.jpg' (640x427x1x3)
[1]: 'image_c1.jpg' (640x427x1x3)

---

# strlen

## Arguments:

- `string1`

## Description:

Return the length of specified string argument.

---

# strlowercase

## Arguments:

- `string`

## Description:

Return a lower-case version of the specified string.

## strreplace

### Arguments:

- `string,search,replace`

### Description:

Search and replace substrings in an input string.

## structuretensors

### Arguments:

- `_scheme={ 0:centered | 1:forward/backward }`

### Description:

Compute the structure tensor field of selected images.

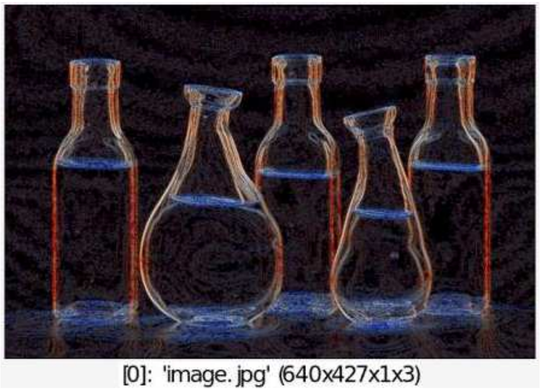
### Default values:

`scheme=0`.

This command has a **tutorial page**.

### Example of use:

```
image.jpg structuretensors abs pow 0.2
```



[0]: 'image.jpg' (640x427x1x3)

## struppercase

### Arguments:

- `string`

### Description:

Return an upper-case version of the specified string.

---

## strvar

### Arguments:

- `"string"`

### Description:

Return a simplified version of the specified string, that can be used as a variable name.

(version that creates a lowercase result, no longer than 128 chars).

---

## strver

### Arguments:

- `_version,_prerelease`

### Description:

Return the specified version number of the G'MIC interpreter, as a string.

### Default values:

`version=$_version` and `prerelease=`.

---

## stylize

### Arguments:

- `[style_image],_fidelity_finest,_fidelity_coarsest,_fidelity_smoothness_finest`

### Description:

Transfer colors and textures from specified style image to selected images, using a multi-scale patch-mathing algorithm.

If instant display window[0] is opened, the steps of the image synthesis are displayed on it.

`init_type` can be *`{ 0:best-match | 1:identity | 2:randomized }`*.

## Default values:

`fidelity_finest=0.5`, `fidelity_coarsest=2`, `fidelity_smoothness_finest=3`, `fidelity_smoothness_coarsest=0.5`, `fidelity_chroma=0.1`, `init_type=0`, `init_resolution=16`, `init_max_gradient=0`, `patch_size_analysis=5`, `patch_size_synthesis=5`, `patch_size_synthesis_final=5`, `nb_matches_finest=2`, `nb_matchesc_coarsest=30`, `penalize_repetitions=2`, `matching_precision=2`, `scale_factor=1.85`, `skip_finest_scales=0` and 'image_matching_command'="s c,-3 match_pca[0] [2] b[0,2] xy,0.7 n[0,2] 0,255 n[1,2] 0,200 a[0,1] c a[1,2] c"'.

---

# sub

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

## Description:

Subtract specified value, image or mathematical expression to selected images, or compute the pointwise difference of selected images.

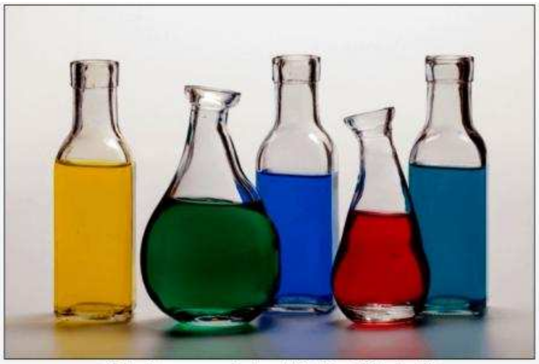
(*equivalent to shortcut command* `-`).

## Examples of use:

### • Example #1

```
image.jpg +sub 30% cut 0,255
```

[0]: 'image.jpg' (640x427x1x3)

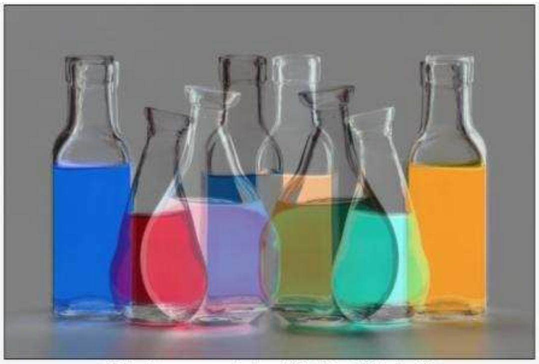


[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +mirror x sub[-1] [0]
```

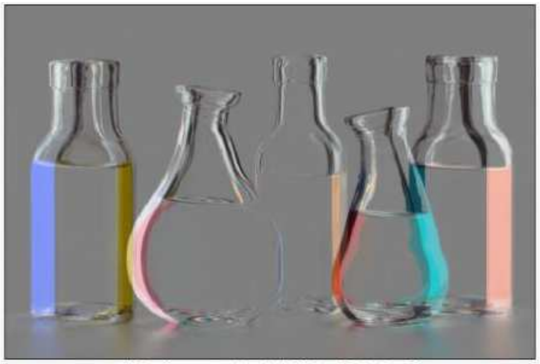
[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

- **Example #3**

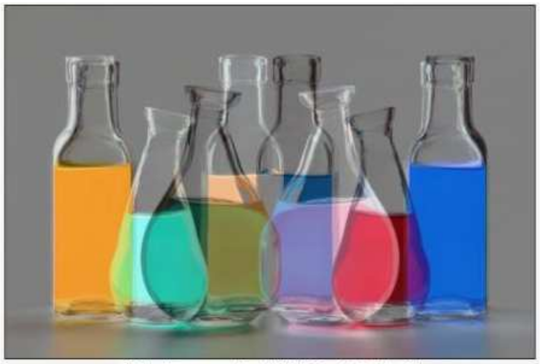
```
image.jpg sub 'i(w/2+0.9*(x-w/2),y)'
```



[0]: 'image.jpg' (640x427x1x3)

- **Example #4**

```
image.jpg +mirror x sub
```



[0]: 'image.jpg' (640x427x1x3)

# sub3d

**Built-in command**

## Arguments:

- `tx,_ty,_tz`

### Description:

Shift selected 3D objects with the opposite of specified displacement vector.

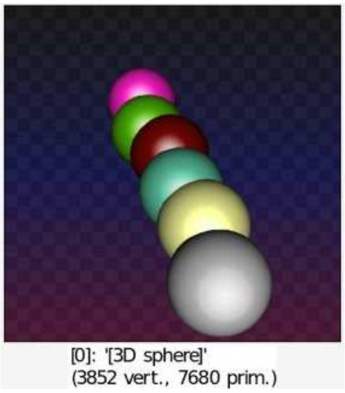
(*equivalent to shortcut command* `3d`).

### Default values:

`ty=tz=0`.

### Example of use:

```
sphere3d 10 repeat 5 { +sub3d[-1] 10,{u(-10,10)},0 color3d[-1] ${-rgb} } add3d
```



[0]: '[3D sphere]'
(3852 vert., 7680 prim.)

---

## sub_alpha

### Arguments:

- `[base_image],0<=_minimize_alpha<=1`

### Description:

Compute the alpha-channel difference (opposite of alpha blending) between the selected images

and the specified base image.
The alpha difference A-B is defined as the image having `minimal` opacity, such that alpha_blend(B,A-B) = A.
The `min_alpha` argument is used to relax the alpha minimality constraint. When set to `1`, alpha is constrained to be minimal. When set to `0`, alpha is maximal (i.e. `255`).

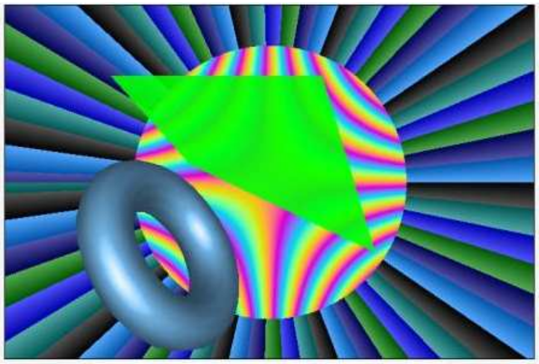
### Default values:

`minimize_alpha=1`.

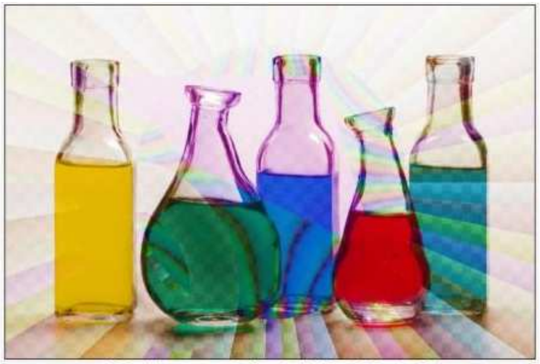
### Example of use:

```
image.jpg testimage2d {w},{h} +sub_alpha[0] [1] display_rgba
```

[0]: 'image.jpg' (640x427x1x3)



[1]: '[2D test image]' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# subdivide3d

### No arguments

## Description:

Subdivide primitives of selected 3D objects.

# superformula3d

## Arguments:

- `resolution>1,m>=1,n1,n2,n3`

## Description:

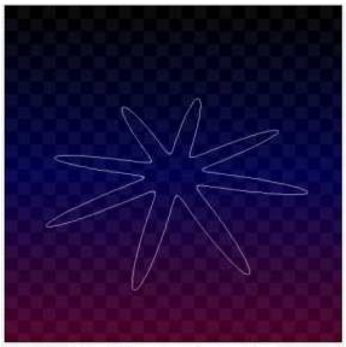
Input 2D superformula curve as a 3D object.

## Default values:

`resolution=1024`, `m=8`, `n1=1`, `n2=5` and `n3=8`.

## Example of use:

```
superformula3d ,
```



[0]: '[3D superformula]'
(1024 vert., 1023 prim.)

## surfels3d

### Arguments:

- `0<=_left_right_attenuation<=1,0<=_top_bottom_attenuation<=1,0<=_closer_furthe`

### Description:

Convert selected images to 3D objects composed of 3D surfels (or 2D edgels for 2D images).

The binary shape is composed of all non-zero voxels.
The resulting 3D object is colored according to the color of non zero voxels.

### Default values:

`left_right_attenuation=1`, `top_bottom_attenuation=1` and `closer_further_attenuation=1`.

### Example of use:

```
100,100,100 = 1,40%,40%,40% = 1,60%,60%,60% distance 1 lt 30% blur 3
gt 50% surfels3d 0.5,0.75,1
```

[0]: '[unnamed]'
(18326 vert., 18324 prim.)

# svd

Built-in command

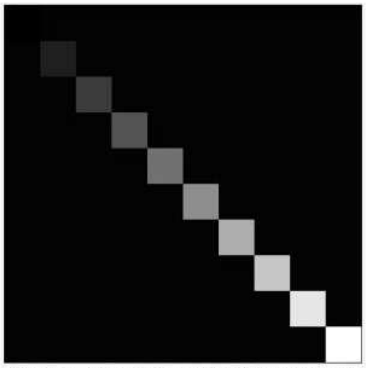
**No arguments**

## Description:

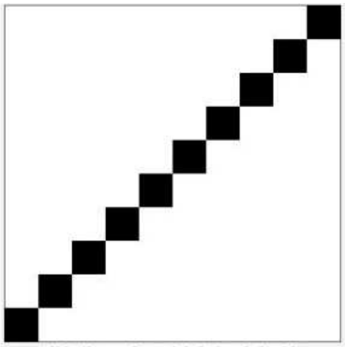
Compute SVD decomposition of selected matrices.

## Example of use:

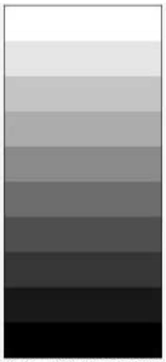
```
10,10,1,1,'x==y?x+u(-0.2,0.2):0' +svd
```



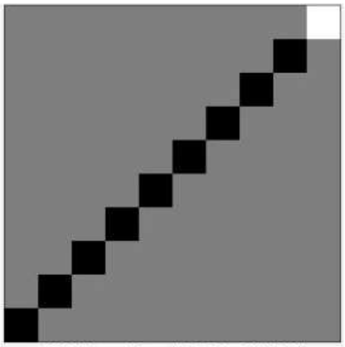
[0]: '[x==y?x+u(-0.2,0.2):0]' (10x10x1x1)



[1]: '[x==y?x+u(-0.2,0_c1.2):0]' (10x10x1x1)



[2]: '[x==y?x+u(-0.2,0_c2.2):0]' (1x10x1x1)



[3]: '[x==y?x+u(-0.2,0_c3.2):0]' (10x10x1x1)

# symmetrize

## Arguments:

- `_x[%],_y[%],_angle,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror },_is_antisymmetry={ 0 | 1 },_swap_sides={ 0 | 1 }`

## Description:

Symmetrize selected images regarding specified axis.

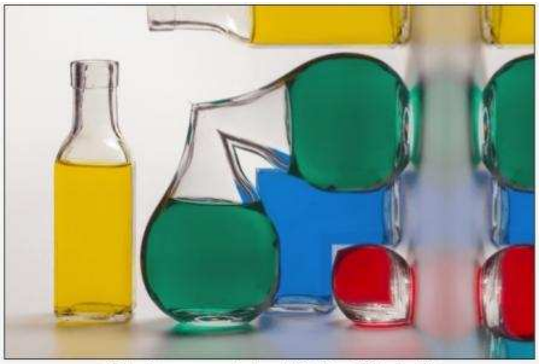
## Default values:

`x=y=50%`, `angle=90`, `boundary_conditions=3`, `is_antisymmetry=0` and `swap_sides=0`.

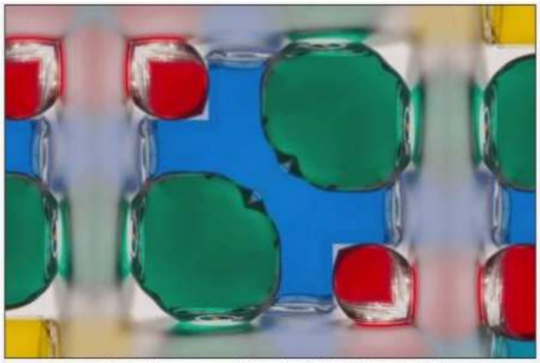
### Example of use:

```
image.jpg +symmetrize 50%,50%,45 +symmetrize[-1] 50%,50%,-45
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c2.jpg' (640x427x1x3)

## syntexturize

### Arguments:

- `_width[%]>0,_height[%]>0`

### Description:

Resynthetize `width'x'height` versions of selected micro-textures by phase randomization.

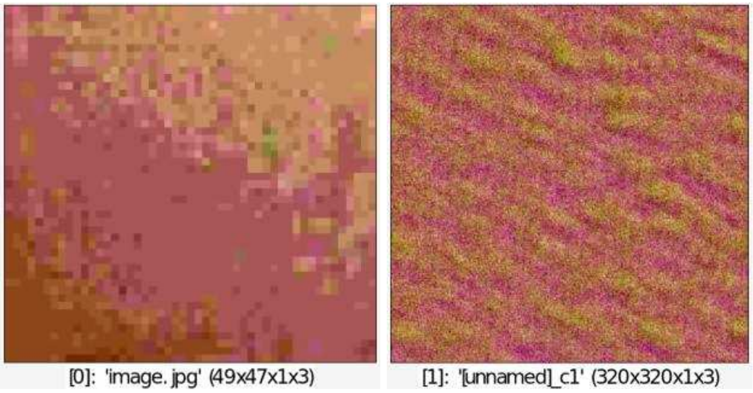
The texture synthesis algorithm is a straightforward implementation of the method described in : **http://www.ipol.im/pub/art/2011/ggm_rpn/**.

### Default values:

`width=height=100%`.

### Example of use:

```
image.jpg crop 2,282,50,328 +syntexturize 320,320
```


[0]: 'image.jpg' (49x47x1x3)
[1]: '[unnamed]_c1' (320x320x1x3)

## syntexturize_matchpatch

### Arguments:

- `_width[%]>0,_height[%]>0,_nb_scales>=0,_patch_size>0,_blending_size>=0,_preci`

### Description:

Resynthetize `width'x'height` versions of selected micro-textures using a patch-matching algorithm.

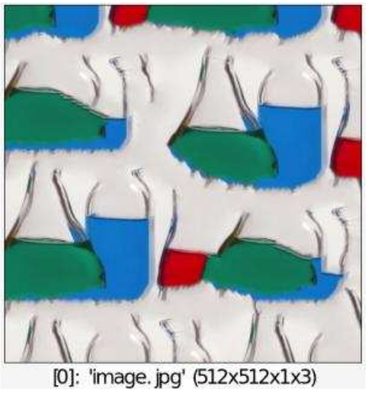
If `nbscales==0`, the number of scales used is estimated from the image size.

### Default values:

`width=height=100%`, `nb_scales=0`, `patch_size=7`, `blending_size=5` and `precision=1`.

### Example of use:

```
image.jpg crop 25%,25%,75%,75% syntexturize_matchpatch 512,512
```



## tan

Built-in command

### No arguments

## Description:

Compute the pointwise tangent of selected images.

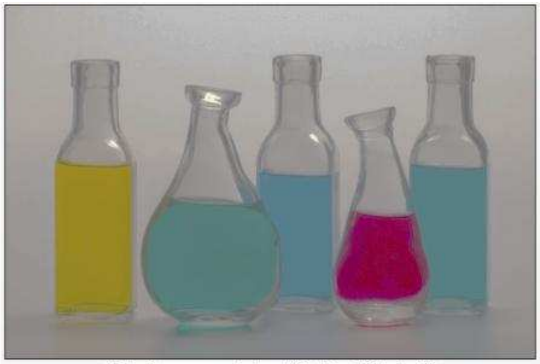
This command has a **tutorial page**.

## Examples of use:

### • Example #1

```
image.jpg +normalize {-0.47*pi},{0.47*pi} tan[-1]
```

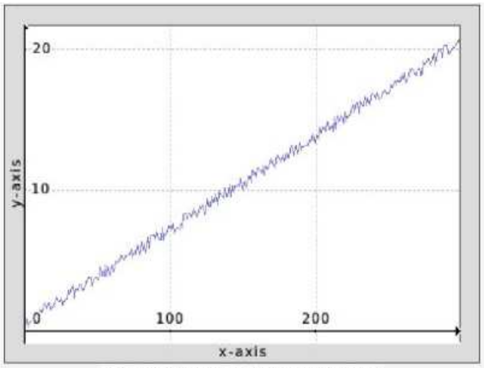
[0]: 'image.jpg' (640x427x1x3)



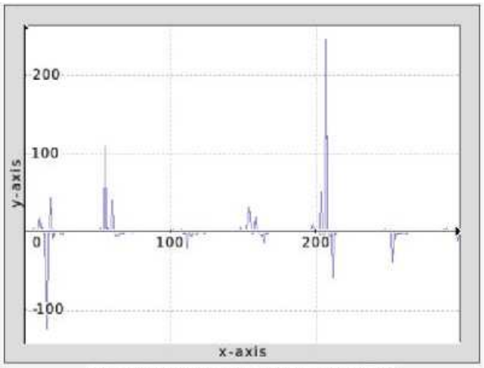
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'20*x/w+u' +tan display_graph 400,300
```



[0]: '[20*x/w+u]' (400x300x1x3)



[1]: '[20*x/w+u]_c1' (400x300x1x3)

---

# tanh

**Built-in command**

### No arguments

## Description:

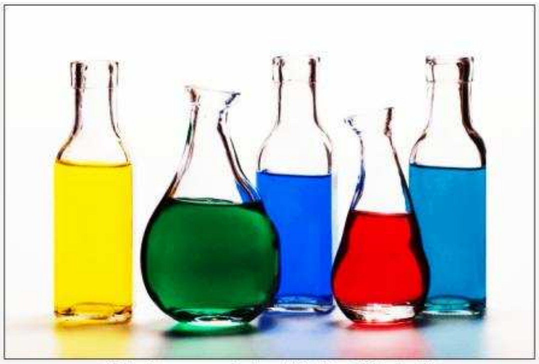
Compute the pointwise hyperbolic tangent of selected images.

## Examples of use:

### • Example #1

```
image.jpg +normalize -3,3 tanh[-1]
```

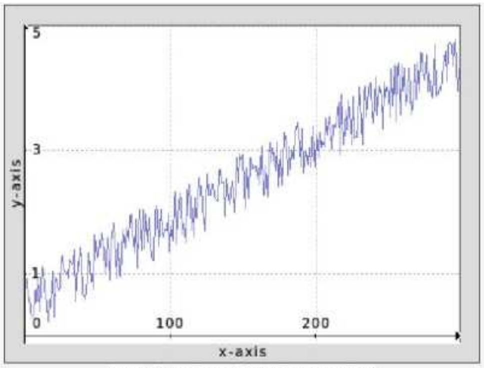
[0]: 'image.jpg' (640x427x1x3)



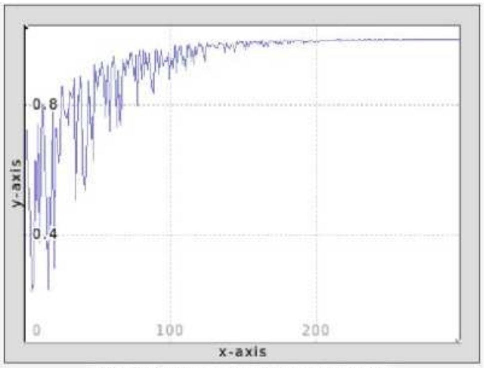
[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
300,1,1,1,'4*x/w+u' +tanh display_graph 400,300
```



[0]: '[4*x/w+u]' (400x300x1x3)



[1]: '[4*x/w+u]_c1' (400x300x1x3)

---

# taquin

## Arguments:

- `M>0,_N>0,_remove_tile={ 0:none | 1:first | 2:last | 3:random },_relief,_border_thickness[%],_border_outline[%],_outline_color`

## Description:

Create MxN taquin puzzle from selected images.

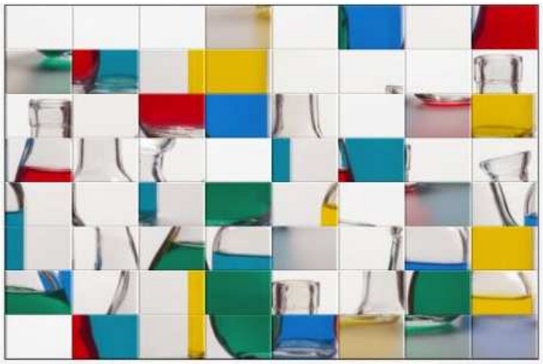
## Default values:

`N=M`, `relief=50`, `border_thickness=5`, `border_outline=0` and `remove_tile=0`.

## Example of use:

```
image.jpg +taquin 8
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x424x1x3)

---

# tensors3d

## Arguments:

- `_radius_factor>=0,_shape={ 0:box | >=N:ellipsoid },_radius_min>=0`

## Description:

Generate 3D tensor fields from selected images.

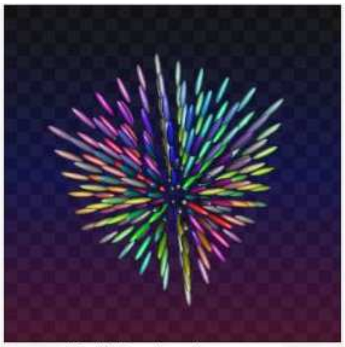
when 'shape'>0, it gives the ellipsoid shape precision.

## Default values:

`radius_factor=1`, `shape=2` and `radius_min=0.05`.

## Example of use:

```
6,6,6,9,"U = [x,y,z] - [w,h,d]/2; U/=norm(U); mul(U,U,3) + 0.3*eye(3)
" tensors3d 0.8
```



[0]: '[3D sphere]'
(9030 vert., 17200 prim.)

---

# testimage2d

## Arguments:

- `_width>0,_height>0,_spectrum>0`

## Description:

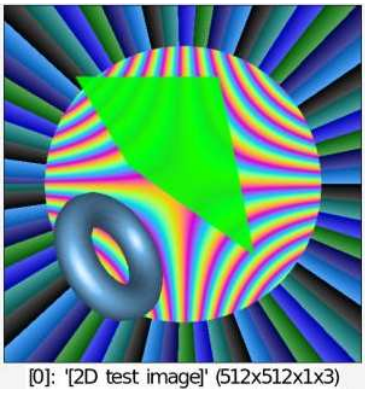
Input a 2D synthetic image.

## Default values:

`width=512`, `height=width` and `spectrum=3`.

## Example of use:

```
testimage2d 512
```



[0]: '[2D test image]' (512x512x1x3)

# tetraedron_shade

## Arguments:

- `x0,y0,z0,x1,y1,z1,x2,y2,z2,x3,y3,z3,R0,G0,B0,...,R1,G1,B1,...,R2,G2,B2,...,R3`

## Description:

Draw tetraedron with interpolated colors on selected (volumetric) images.

# tetris

## Arguments:

- `_scale>0`

## Description:

Apply tetris effect on selected images.

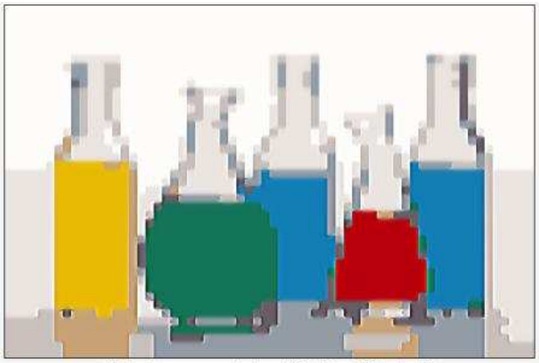
## Default values:

`scale=10`.

## Example of use:

```
image.jpg +tetris 10
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

---

# text

Built-in command

## Arguments:

- `text,_x[%|~],_y[%|~],_{ font_height[%]>=0 | custom_font },_opacity,_color1,...`

## Description:

Draw specified colored text string on selected images.

(*equivalent to shortcut command* `t`).

If one of the x or y argument ends with a `~`, its value is expected to be a centering ratio (in [0,1]) rather than a position.
Usual centering ratio are ***{ 0:left-justified | 0.5:centered | 1:right-justified }***.
Sizes `13` and `128` are special and correspond to binary fonts (no-antialiasing). Any other font size is rendered with anti-aliasing.
Specifying an empty target image resizes it to new dimensions such that the image contains the entire text string.
A custom font can be specified as a variable name that stores an image list of 256 or 512 items (512 for 256 character sprites + 256 associated opacities), or as an image selection that is a serialized version of such an image list.

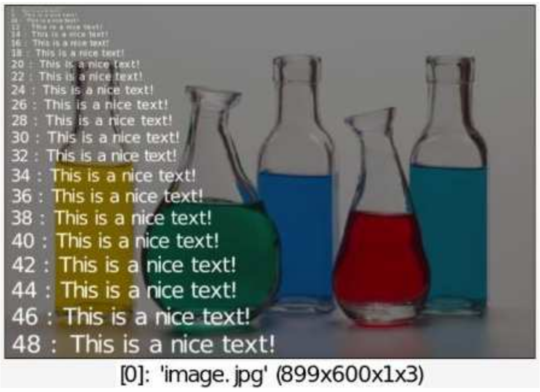
## Default values:

`x=y=0.01~`, `font_height=16`, `opacity=1` and `color1=0`.

## Examples of use:

- **Example #1**

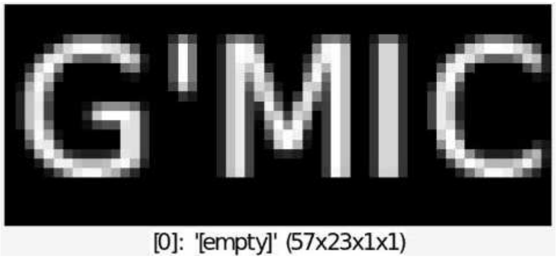
```
image.jpg rescale2d ,600 div 2 y=0 repeat 30 { text {2*$>}" : This is
a nice text!",10,$y,{2*$>},0.9,255 y+={2*$>} }
```


18 : This is a nice text!
20 : This is a nice text!
22 : This is a nice text!
24 : This is a nice text!
26 : This is a nice text!
28 : This is a nice text!
30 : This is a nice text!
32 : This is a nice text!
34 : This is a nice text!
36 : This is a nice text!
38 : This is a nice text!
40 : This is a nice text!
42 : This is a nice text!
44 : This is a nice text!
46 : This is a nice text!
48 : This is a nice text!

[0]: 'image.jpg' (899x600x1x3)

- **Example #2**

```
0 text "G'MIC",0,0,23,1,255
```



[0]: '[empty]' (57x23x1x1)

# text3d

## Arguments:

- `text,_{ font_height>=0 | custom_font },_depth>0,_smoothness`

## Description:

Input a 3D text object from specified text.

## Default values:

`font_height=53`, `depth=10` and `smoothness=1.5`.

## Example of use:

```
text3d "G'MIC as a\n3D logo!"
```

G'MIC as a
3D logo!

[0]: '[3D text 'G'MIC as a 3D logo!']' (32100 vert., 64160 prim.)

---

# text_outline

## Arguments:

- `text,_x[%|~],_y[%|~],{ _font_height[%]>0 | custom_font },_outline>=0,_opacity,_color1,...`

## Description:

Draw specified colored and outlined text string on selected images.

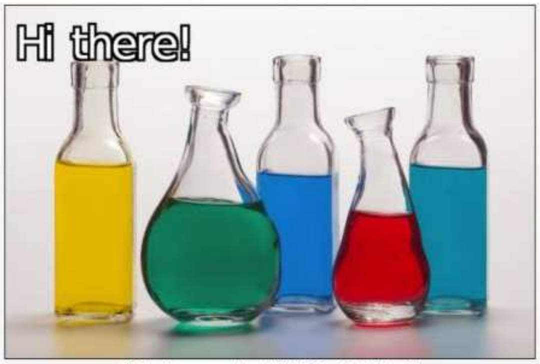
If one of the x or y argument ends with a `~`, its value is expected to be
a centering ratio (in [0,1]) rather than a position.
Usual centering ratio are ***{ 0:left-justified | 0.5:centered | 1:right-justified }***.

## Default values:

`x=y=0.01~`, `font_height=7.5%`, `outline=2`, `opacity=1`, `color1=color2=color3=255` and `color4=255`.

## Example of use:

```
image.jpg text_outline "Hi there!",10,10,63,3
```



[0]: 'image.jpg' (640x427x1x3)

---

# text_pointcloud3d

## Arguments:

- `_"text1",_"text2",_smoothness`

## Description:

Input 3D text pointcloud from the two specified strings.

## Default values:

`text1="text1"`, `text2="text2"` and `smoothness=1`.

## Example of use:

```
text_pointcloud3d "G'MIC","Rocks!"
```

[0]: '[3D text pointcloud]'
(16038 vert., 30528 prim.)

---

# texturize3d

## Arguments:

- `[ind_texture],_[ind_coords]`

## Description:

Texturize selected 3D objects with specified texture and coordinates.

(*equivalent to shortcut command* `t3d`).

When `[ind_coords]` is omitted, default XY texture projection is performed.

## Default values:

`ind_coords=(undefined)`.

## Example of use:

```
image.jpg torus3d 100,30 texturize3d[-1] [-2] keep[-1]
```

[0]: '[3D torus]' (288 vert., 288 prim.)

# texturize_canvas

## Arguments:

- `_amplitude>=0,_fibrousness>=0,_emboss_level>=0`

## Description:

Add paint canvas texture to selected images.

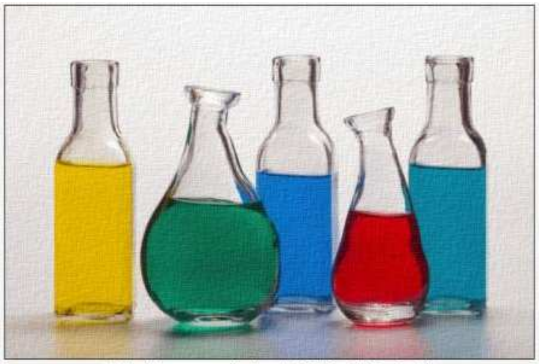
## Default values:

`amplitude=20`, `fibrousness=3` and `emboss_level=0.6`.

## Example of use:

```
image.jpg +texturize_canvas ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# texturize_paper

**No arguments**

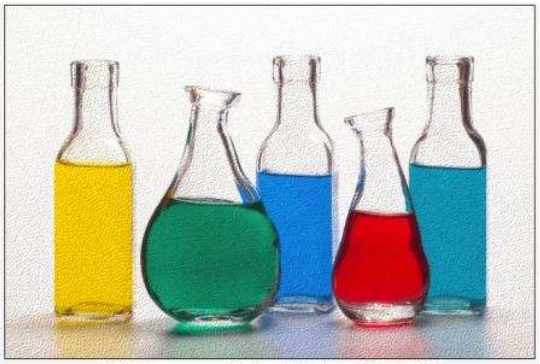
### Description:

Add paper texture to selected images.

### Example of use:

```
image.jpg +texturize_paper
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c2.jpg' (640x427x1x3)

## thickline

### Arguments:

- `x0[%],y0[%],x1[%],y1[%],_thickness,_opacity,_color1`

### Description:

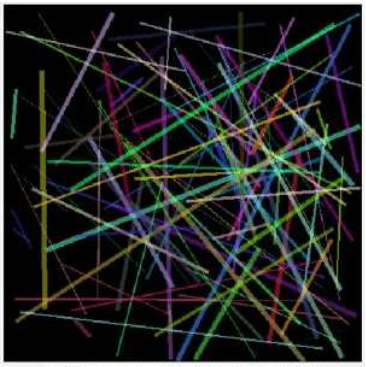
Draw specified colored thick line on selected images.

### Default values:

`thickness=2`, `opacity=1` and `color1=0`.

### Example of use:

```
400,400,1,3 repeat 100 thickline {u([w,h,w,h,5])},0.5,${-rgb} done
```



[0]: '[unnamed]' (400x400x1x3)

# thickspline

## Arguments:

- `x0[%],y0[%],u0[%],v0[%],x1[%],y1[%],u1[%],v1[%],_thickness,_opacity,_color1,...`

## Description:

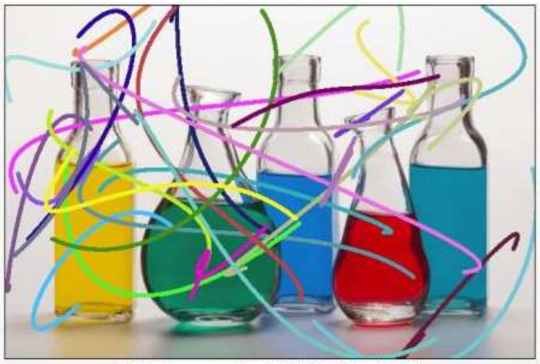
Draw specified colored thick spline curve on selected images (cubic hermite spline).

## Default values:

`thickness=3`, `opacity=1` and `color1=0`.

## Example of use:

```
image.jpg repeat 30 { thickspline {u(100)}%,{u(100)}%,{u(-600,600)},{u(-600,600)},{u(100)}%,{u(100)}%,{u(-600,600)},{u(-600,600)},3,1,${-RGB} }
```



[0]: 'image.jpg' (640x427x1x3)

# thinning

## Arguments:

- `_boundary_conditions={ 0:dirichlet | 1:neumann }`

## Description:

Compute skeleton of binary shapes using morphological thinning

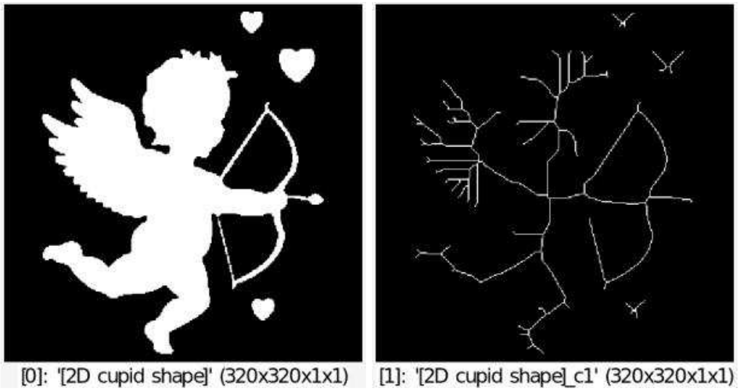
(beware, this is a quite slow iterative process)

## Default values:

`boundary_conditions=1`.

## Example of use:

```
shape_cupid 320 +thinning
```


[0]: '[2D cupid shape]' (320x320x1x1)
[1]: '[2D cupid shape]_c1' (320x320x1x1)

# threshold

## Arguments:

- `value[%],_is_soft={ 0 | 1 } :`

## Description:

Threshold values of selected images.

`soft` can be ***{ 0:hard-thresholding | 1:soft-thresholding }***.

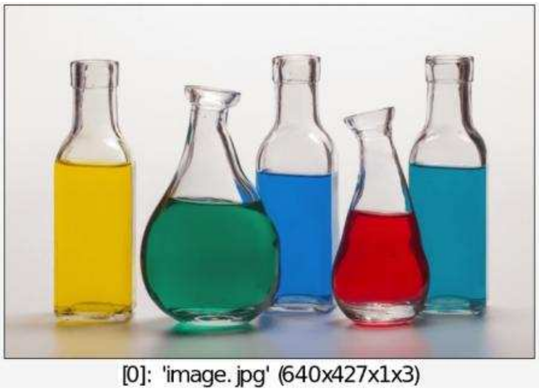
## Default values:

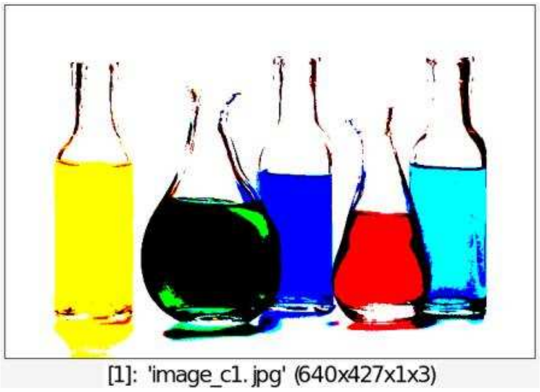
`is_soft=0`.

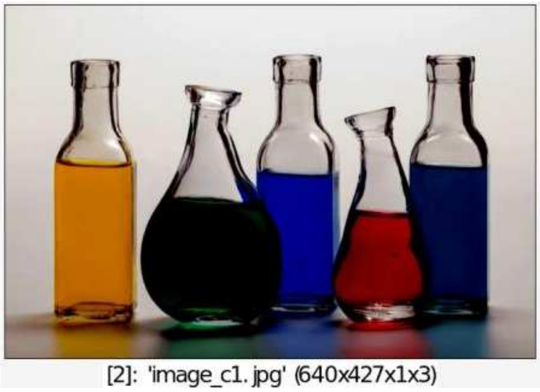
This command has a **tutorial page**.

## Example of use:

```
image.jpg +threshold[0] 50% +threshold[0] 50%,1
```







---

# tic

**No arguments**

## Description:

Initialize tic-toc timer.

Use it in conjunction with `toc`.

---

# tixy

## Arguments:

- `"expression"`

## Description:

Animate specified mathematical expression with a 16x16 grid of circles, using the rules described at **https://tixy.land**.

---

# to_a

### No arguments

### Description:

Force selected images to have an alpha channel.

---

# to_automode

### No arguments

### Description:

Force selected images to be in the most significant color mode.

This commands checks for useless alpha channel (all values equal to 255), as well as detects grayscale images encoded as color images.

---

# to_color

### No arguments

### Description:

Force selected images to be in color mode (RGB or RGBA).

---

# to_colormode

### Arguments:

- `mode={ 0:adaptive | 1:G | 2:GA | 3:RGB | 4:RGBA }`

### Description:

Force selected images to be in a given color mode.

### Default values:

`mode=0`.

---

# to_gray

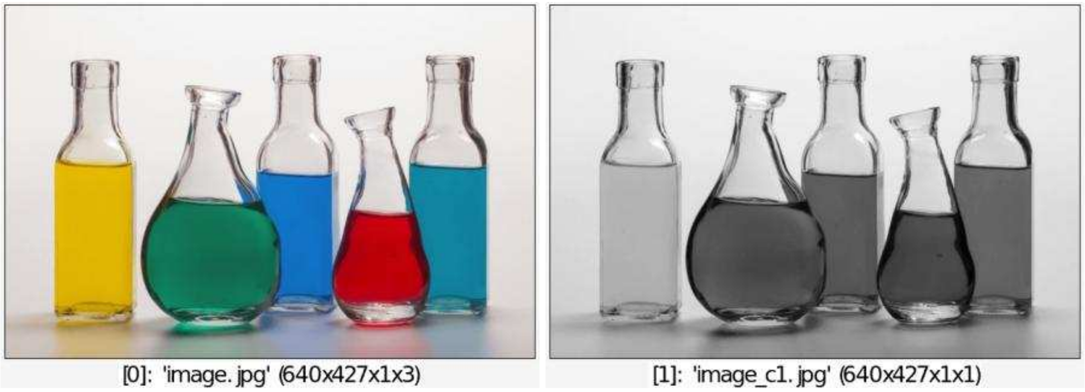
### No arguments

### Description:

Force selected images to be in GRAY mode.

### Example of use:

```
image.jpg +to_gray
```



[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x1)

## to_graya

**No arguments**

### Description:

Force selected images to be in GRAYA mode.

## to_pseudogray

### Arguments:

- `_max_step>=0,_is_perceptual_constraint={ 0 | 1 },_bits_depth>0`

### Description:

Convert selected scalar images ([0-255]-valued) to pseudo-gray color images.

### Default values:

`max_step=5`, `is_perceptual_constraint=1` and `bits_depth=8`.

The original pseudo-gray technique has been introduced by Rich Franzen **http://r0k.us/graphics/pseudoGrey.html**.
Extension of this technique to arbitrary increments for more tones, has been done by David Tschumperlé.

# to_rgb

### No arguments

## Description:

Force selected images to be in RGB mode.

# to_rgba

### No arguments

## Description:

Force selected images to be in RGBA mode.

# toc

### No arguments

## Description:

Display elapsed time of the tic-toc timer since the last call to `tic`.

This command returns the elapsed time in the status value.
Use it in conjunction with `tic`.

# tones

## Arguments:

- `N>0`

## Description:

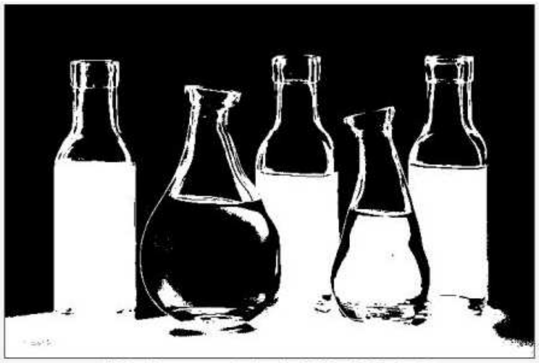
Get N tones masks from selected images.

## Example of use:

```
image.jpg +tones 3
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x1)



[2]: 'image_c2.jpg' (640x427x1x1)

[3]: 'image_c2.jpg' (640x427x1x1)

# topographic_map

## Arguments:

- `_nb_levels>0,_smoothness`

## Description:

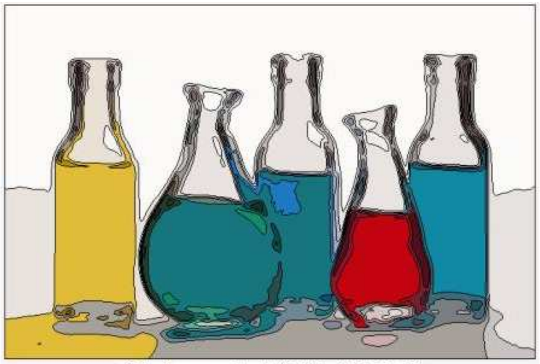
Render selected images as topographic maps.

## Default values:

`nb_levels=16` and `smoothness=2`.

## Example of use:

```
image.jpg topographic_map 10
```



[0]: 'image.jpg' (640x427x1x3)

# torus3d

## Arguments:

- `_radius1,_radius2,_nb_subdivisions1>2,_nb_subdivisions2>2`

## Description:

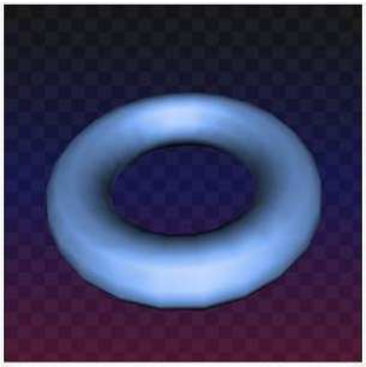
Input 3D torus at (0,0,0), with specified geometry.

## Default values:

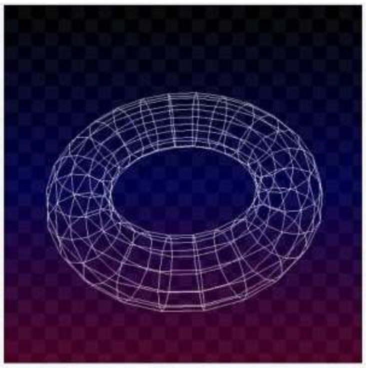
`radius1=1`, `radius2=0.3`, `nb_subdivisions1=24` and `nb_subdivisions2=12`.

## Example of use:

```
torus3d 10,3 +primitives3d 1 color3d[-2] ${-rgb}
```



[0]: '[3D torus]' (288 vert., 288 prim.)



[1]: '[3D torus]_c1' (288 vert., 576 prim.)

# transform_polar

## Arguments:

- `"expr_radius","expr_angle",_center_x[%],_center_y[%],_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Apply user-defined transform on polar representation of selected images.

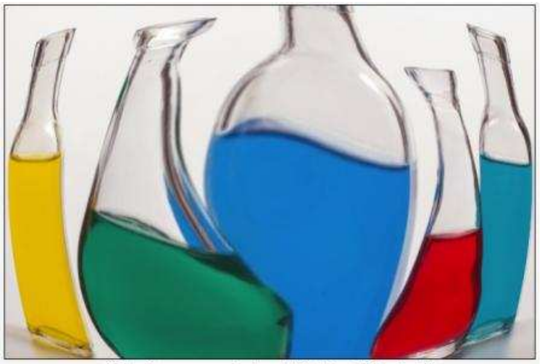
## Default values:

`expr_radius=R-r`, `expr_rangle=a`, `center_x=center_y=50%` and `boundary_conditions=3`.

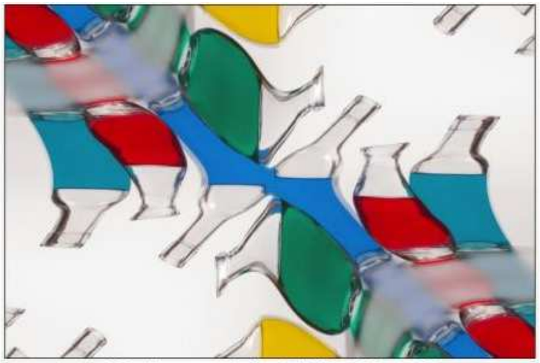
## Example of use:

```
image.jpg +transform_polar[0] R*(r/R)^2,a +transform_polar[0] r,2*a
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)



[2]: 'image_c1.jpg' (640x427x1x3)

# transition

## Arguments:

- `[transition_shape],nb_added_frames>=0,100>=shading>=0,_single_frame_only={ -1=disabled | >=0 }`

## Description:

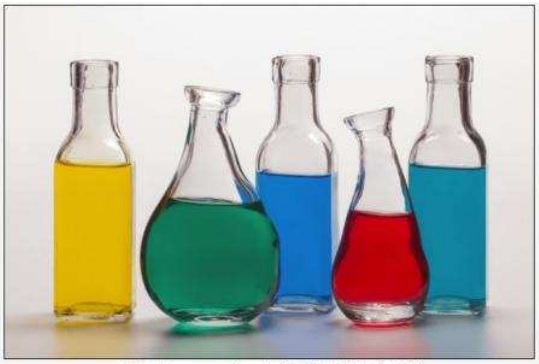
Generate a transition sequence between selected images.

## Default values:

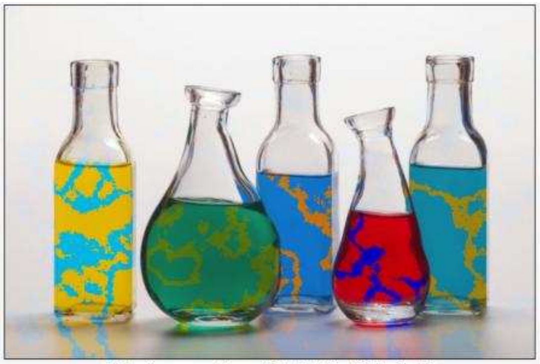
`shading=0` and `single_frame_only=-1`.

**Example of use:**

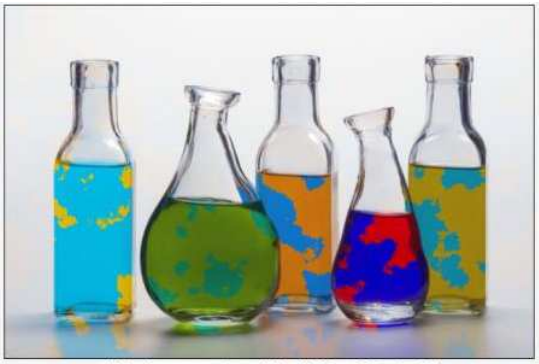
```
image.jpg +mirror c 100%,100% plasma[-1] 1,1,6 transition[0,1] [2],5
```



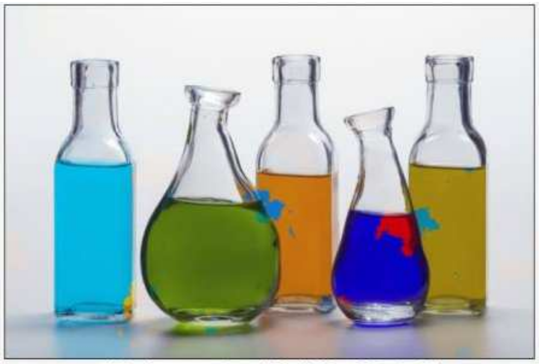
[0]: 'image. jpg' (640x427x1x3)



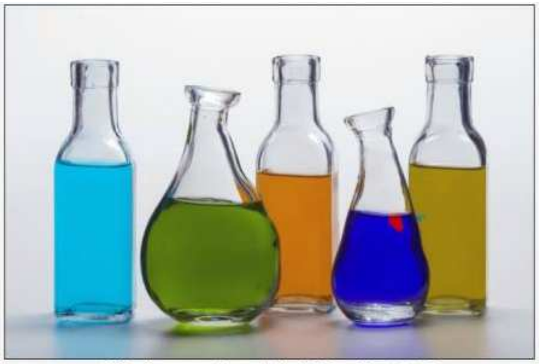
[1]: 'image. jpg #1' (640x427x1x3)



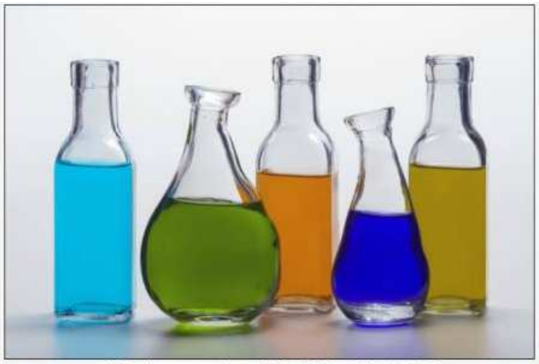
[2]: 'image. jpg #2' (640x427x1x3)



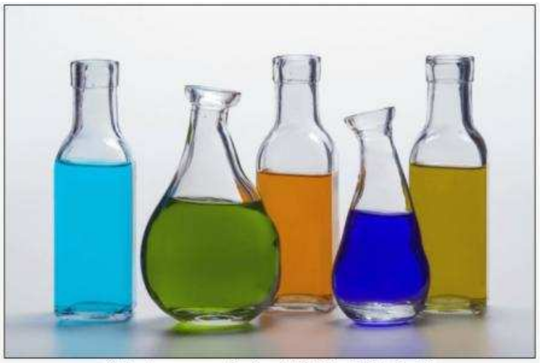
[3]: 'image. jpg #3' (640x427x1x3)



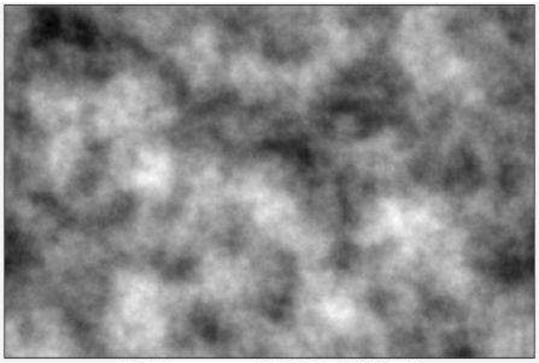
[4]: 'image. jpg #4' (640x427x1x3)



[5]: 'image. jpg #5' (640x427x1x3)



[6]: 'image_c1. jpg' (640x427x1x3)



[7]: '[unnamed]' (640x427x1x1)

## transition3d

## Arguments:

- `_nb_frames>=2,_nb_xtiles>0,_nb_ytiles>0,_axis_x,_axis_y,_axis_z,_is_antialias 0 | 1 }`

## Description:

Create 3D transition sequence between selected consecutive images.

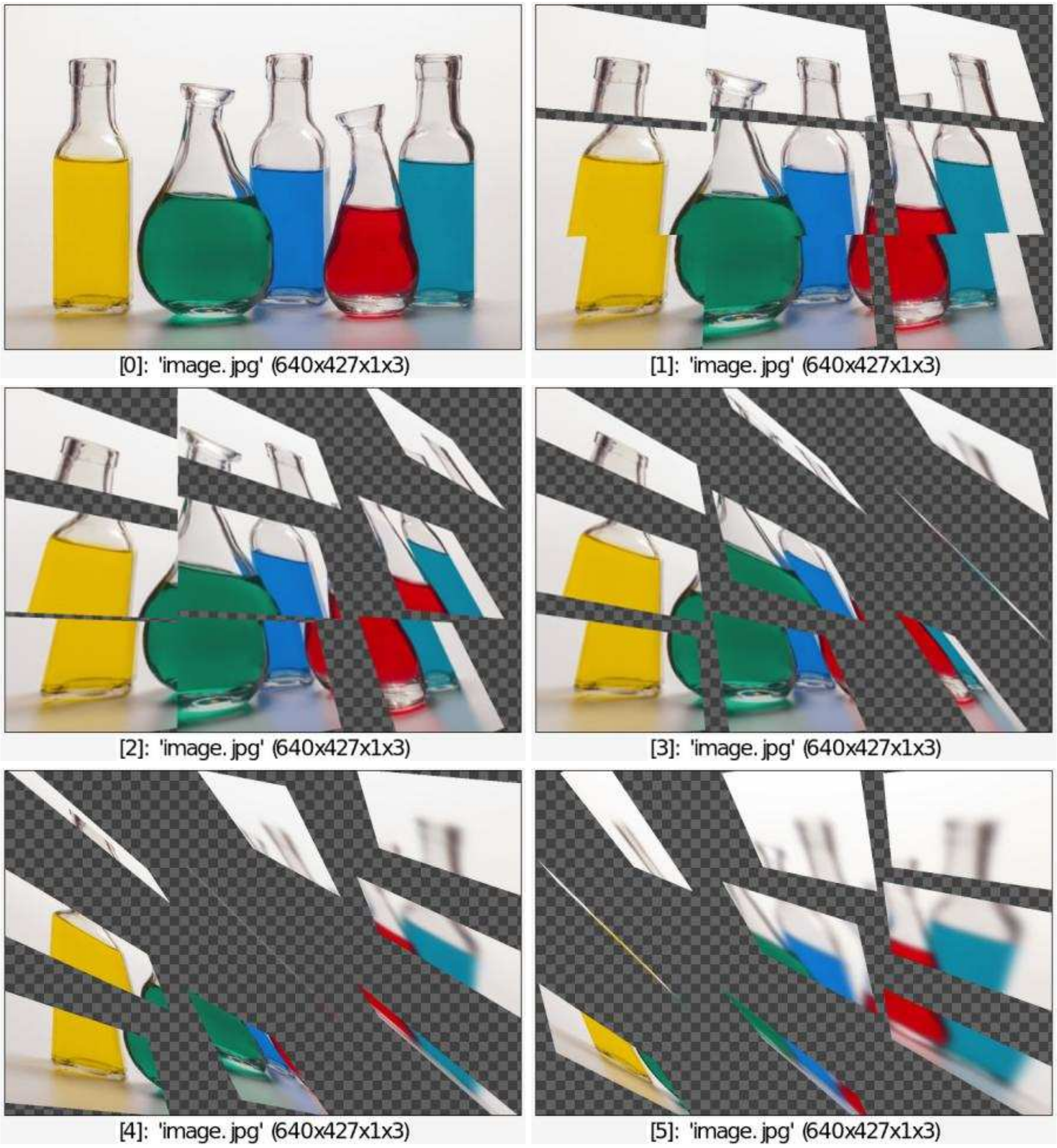
`axis_x`, `axis_y` and `axis_z` can be set as mathematical expressions, depending on `x` and `y`.

## Default values:

`nb_frames=10`, `nb_xtiles=nb_ytiles=3`, `axis_x=1`, `axis_y=1`, `axis_z=0` and `is_antialias=1`.

## Example of use:

```
image.jpg +blur 5 transition3d 9 display_rgba
```



[0]: 'image.jpg' (640x427x1x3)

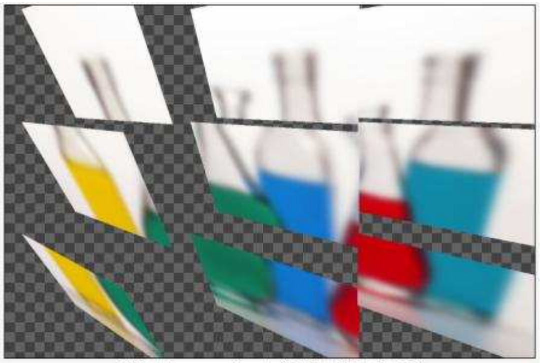
[1]: 'image.jpg' (640x427x1x3)

[2]: 'image.jpg' (640x427x1x3)

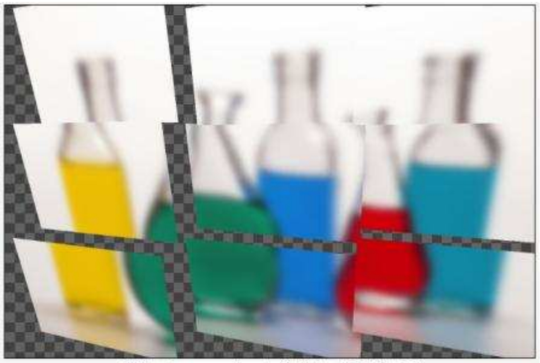
[3]: 'image.jpg' (640x427x1x3)

[4]: 'image.jpg' (640x427x1x3)

[5]: 'image.jpg' (640x427x1x3)



[6]: 'image.jpg' (640x427x1x3)



[7]: 'image.jpg' (640x427x1x3)

[8]: 'image_c1.jpg' (640x427x1x3)

# transpose

**No arguments**

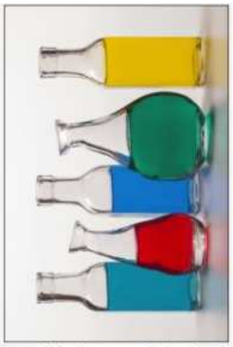
## Description:

Transpose selected matrices.

## Example of use:

```
image.jpg +transpose
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (427x640x1x3)

# triangle3d

## Arguments:

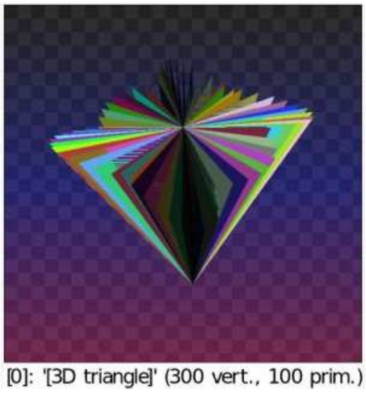
- `x0,y0,z0,x1,y1,z1,x2,y2,z2`

## Description:

Input 3D triangle at specified coordinates.

## Example of use:

```
repeat 100 { a:=$>*pi/50 triangle3d 0,0,0,0,0,3,{cos(3*$a)},
{sin(2*$a)},0 color3d[-1] ${-rgb} } add3d
```



[0]: '[3D triangle]' (300 vert., 100 prim.)

# triangle_shade

## Arguments:

- `x0,y0,x1,y1,x2,y2,R0,G0,B0,...,R1,G1,B1,...,R2,G2,B2,...`

## Description:

Draw triangle with interpolated colors on selected images.

## Example of use:

```
image.jpg triangle_shade
20,20,400,100,120,200,255,0,0,0,255,0,0,0,255
```

[0]: 'image.jpg' (640x427x1x3)

---

## trisolve

**Built-in command**

### Arguments:

- `[image]`

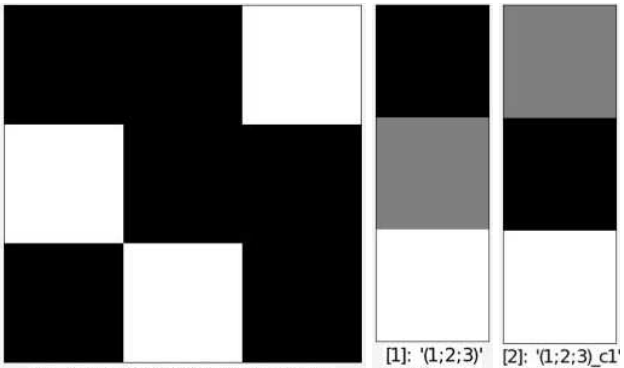
### Description:

Solve tridiagonal system AX = B for selected B-vectors and specified tridiagonal A-matrix.

Tridiagonal matrix must be stored as a 3 column vector, where 2nd column contains the diagonal coefficients, while 1st and 3rd columns contain the left and right coefficients.

### Example of use:

```
(0,0,1;1,0,0;0,1,0) (1;2;3) +trisolve[-1] [-2]
```



[0]: '(0,0,1;1,0,0;0,1,0)' (3x3x1x1)

[1]: '(1;2;3)' (1x3x1x1)

[2]: '(1;2;3)_c1' (1x3x1x1)

---

## truchet

### Arguments:

- `_scale>0,_radius>=0,_pattern_type={ 0:straight | 1:curved }`

### Description:

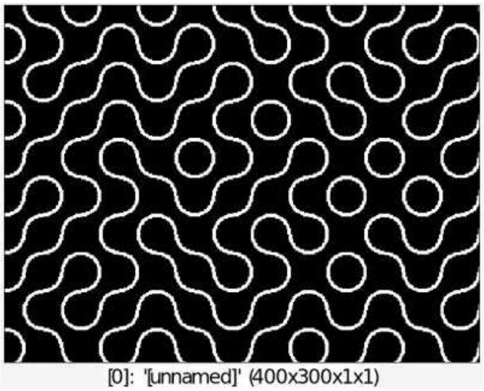
Fill selected images with random truchet patterns.

## Default values:

`scale=32`, `radius=5` and `pattern_type=1`.

## Example of use:

```
400,300 truchet ,
```



[0]: '[unnamed]' (400x300x1x1)

---

# tsp

## Arguments:

- `_precision>=0`

## Description:

Try to solve the 'travelling salesman' problem, using a combination of greedy search and 2-opt algorithms.

Selected images must have dimensions Nx1x1xC to represent N cities each with C-dimensional coordinates.
This command re-order the selected data along the x-axis so that the point sequence becomes a shortest path.

## Default values:

`precision=256`.

## Example of use:

```
256,1,1,2 rand 0,512 tsp , 512,512,1,3 repeat w#0 circle[-1]
{0,I[$>]},2,1,255,255,255 line[-1] {0,boundary=2;[I[$>],I[$>+1]]},
1,255,128,0 done keep[-1]
```

[0]: '[unnamed]' (512x512x1x3)

# tunnel

## Arguments:

- `_level>=0,_factor>0,_centering_x,_centering_y,_opacity,_angle`

## Description:

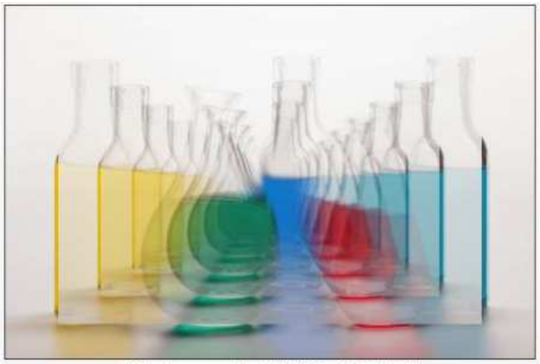
Apply tunnel effect on selected images.

## Default values:

`level=9`, `factor=80%`, `centering_x=centering_y=0.5`, `opacity=1` and `angle=0`

## Example of use:

```
image.jpg tunnel 20
```



[0]: 'image.jpg' (640x427x1x3)

# turbulence

## Arguments:

- `_radius>0,_octaves={1,2,3...,12},_alpha>0,_difference={-10,10},_mode={0,1,2,3}`

## Description:

Render fractal noise or turbulence on selected images.

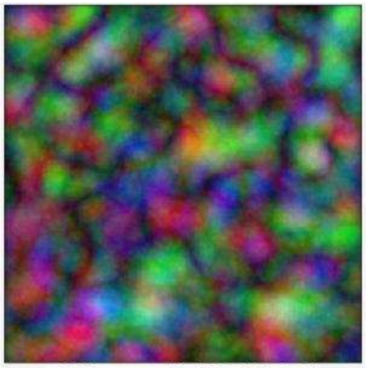
## Default values:

`radius=32`, `octaves=6`, `alpha=3`, `difference=0` and `mode=0`.

This command has a **tutorial page**.

## Example of use:

```
400,400,1,3 turbulence 16
```



[0]: '[unnamed]' (400x400x1x3)

# tv_flow

## Arguments:

- `_nb_iter>=0,_dt,_keep_sequence={ 0 | 1 }`

## Description:

Apply iterations of the total variation flow on selected images.

## Default values:

`nb_iter=10`, `dt=30` and `keep_sequence=0`.

## Example of use:

```
image.jpg +tv_flow 40
```

[0]: 'image.jpg' (640x427x1x3)

[1]: 'image_c1.jpg' (640x427x1x3)

---

# twirl

## Arguments:

- `_amplitude,_center_x[%],_center_y[%],_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

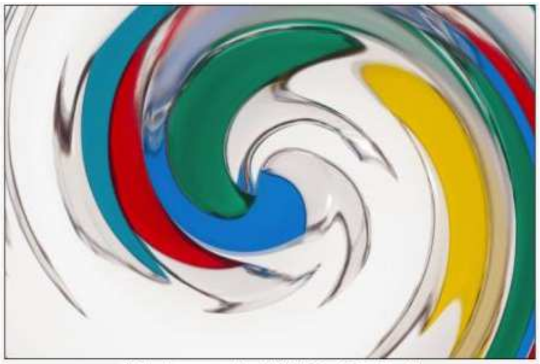
Apply twirl deformation on selected images.

## Default values:

`amplitude=1`, `center_x=center_y=50%` and `boundary_conditions=3`.

## Example of use:

```
image.jpg twirl 0.6
```


[0]: 'image.jpg' (640x427x1x3)

---

# uint82base64

## Arguments:

- `_encoding={ 0:base64 | 1:base64url }`

### Description:

Encode the values of the latest of the selected images as a base64-encoded string.

The string can be decoded using command `base642uint8`.
Selected images must have values that are integers in [0,255].

### Default values:

`encoding=0`.

---

## uncommand

**Built-in command**

### Arguments:

- `command_name[,_command_name2,...]` or
- `*`

### Description:

Discard definition of specified custom commands.

Set argument to `*` for discarding all existing custom commands.

(*equivalent to shortcut command* `um`).

---

## undistort

### Arguments:

- `-1<=_amplitude<=1,_aspect_ratio,_zoom,_center_x[%],_center_y[%],_boundary_cor`

### Description:

Correct barrel/pincushion distortions occurring with wide-angle lens.

References:
[1] Zhang Z. (1999). Flexible camera calibration by viewing a plane from unknown orientation.
[2] Andrew W. Fitzgibbon (2001). Simultaneous linear estimation of multiple view geometry and lens distortion.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

### Default values:

`amplitude=0.25`, `aspect_ratio=0`, `zoom=0`, `center_x=center_y=50%` and `boundary_conditions=0`.

# uniform_distribution

## Arguments:

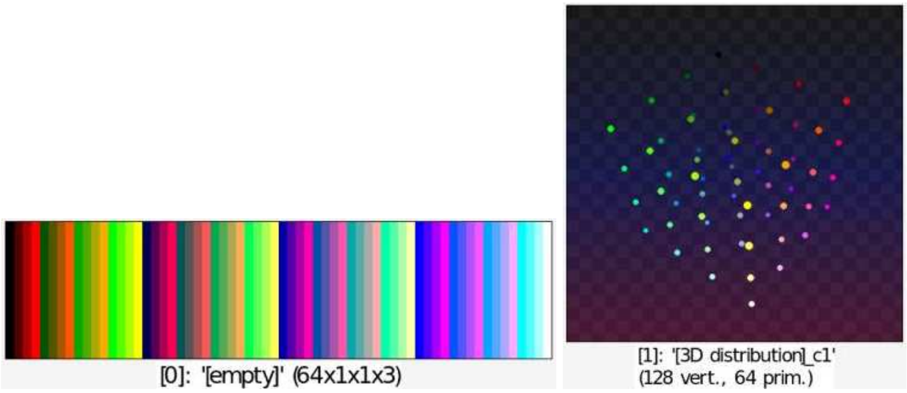
- `nb_levels>=1,spectrum>=1`

## Description:

Input set of uniformly distributed spectrum-d points in [0,1]^spectrum.

## Example of use:

```
uniform_distribution 64,3 * 255 +distribution3d circles3d[-1] 10
```



# unroll

Built-in command

## Arguments:

- `_axis={ x | y | z | c }`

## Description:

Unroll selected images along specified axis.

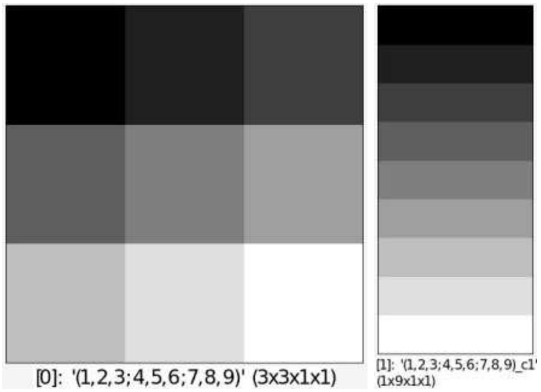
(*equivalent to shortcut command* `y`).

## Default values:

`axis=y`.

## Example of use:

```
(1,2,3;4,5,6;7,8,9) +unroll y
```


[0]: '(1,2,3;4,5,6;7,8,9)' (3x3x1x1)
[1]: '(1,2,3;4,5,6;7,8,9)_c1'
(1x9x1x1)

---

## unserialize

Built-in command

**No arguments**

### Description:

Recreate lists of images from serialized image buffers, obtained with command `serialize`.

---

## unsharp

### Arguments:

- `radius[%]>=0,_amount>=0,_threshold[%]>=0`

### Description:

Apply unsharp mask on selected images.

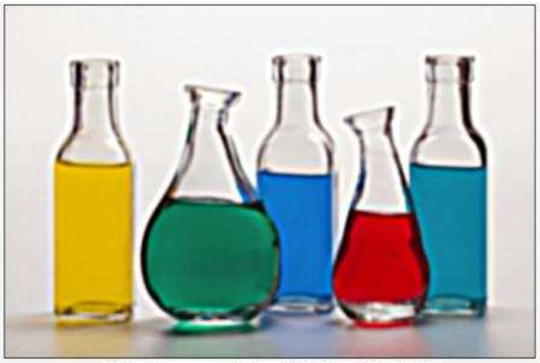
### Default values:

`amount=2` and `threshold=0`.

### Example of use:

```
image.jpg blur 3 +unsharp 1.5,15 cut 0,255
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## unsharp_octave

### Arguments:

- `_nb_scales>0,_radius[%]>=0,_amount>=0,threshold[%]>=0`

### Description:

Apply octave sharpening on selected images.

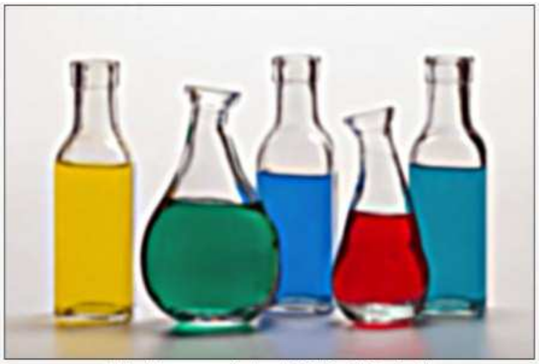
### Default values:

`nb_scales=4`, `radius=1`, `amount=2` and `threshold=0`.

### Example of use:

```
image.jpg blur 3 +unsharp_octave 4,5,15 cut 0,255
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

## update

**No arguments**

### Description:

Update commands from the latest definition file on the G'MIC server.

(*equivalent to shortcut command* `up`).

# upscale_smart

## Arguments:

- `width[%],_height[%],_depth,_smoothness>=0,_anisotropy=[0,1],sharpening>=0`

## Description:

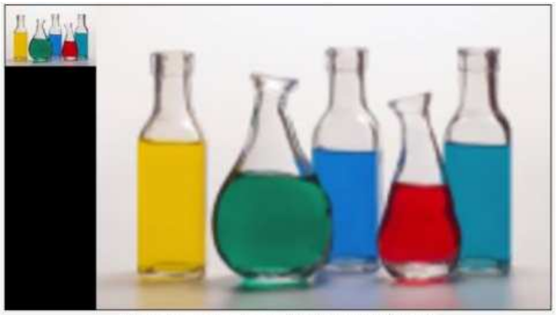
Upscale selected images with an edge-preserving algorithm.

## Default values:

`height=100%`, `depth=100%`, `smoothness=2`, `anisotropy=0.4` and `sharpening=10`.

## Example of use:

```
image.jpg rescale2d ,100 +upscale_smart 500%,500% append x
```



[0]: 'image.jpg' (900x500x1x3)

# vanvliet

**Built-in command**

## Arguments:

- `std_deviation>=0[%],order={ 0 | 1 | 2 | 3 },axis={ x | y | z | c },_boundary_conditions`

## Description:

Apply Vanvliet recursive filter on selected images, along specified axis and with

specified standard deviation, order and boundary conditions.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

## Default values:

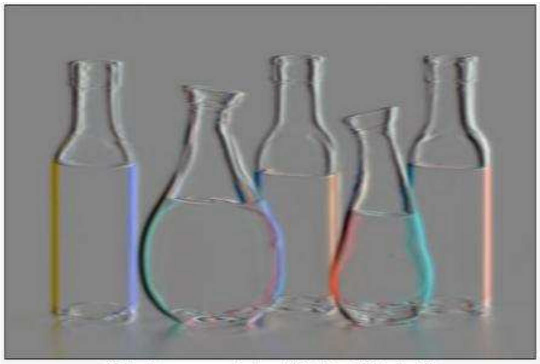
`boundary_conditions=1`.

## Examples of use:

### • Example #1

```
image.jpg +vanvliet 3,1,x
```

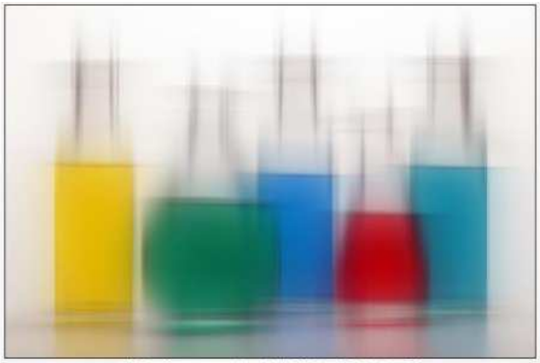
[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

### • Example #2

```
image.jpg +vanvliet 30,0,x vanvliet[-2] 30,0,y add
```



[0]: 'image.jpg' (640x427x1x3)

---

# variance_patch

## Arguments:

- `_patch_size>=1`

## Description:

Compute variance of each images patch centered at (x,y), in selected images.

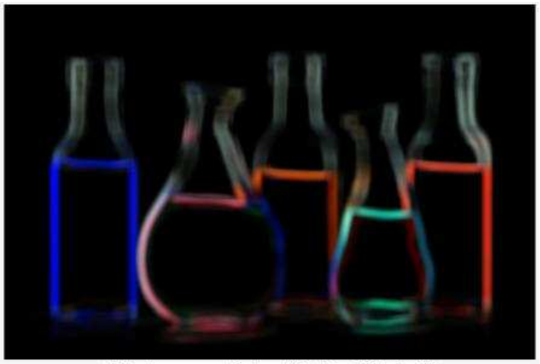
## Default values:

`patch_size=16`

### Example of use:

```
image.jpg +variance_patch
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c2.jpg' (640x427x1x3)

---

## vector2tensor

**No arguments**

### Description:

Convert selected vector fields to corresponding tensor fields.

---

## verbose

**Built-in command**

### Arguments:

- `level` or
- `{ + | - }`

### Description:

Set or increment/decrement the verbosity level. Default level is 0.

(*equivalent to shortcut command* `v`).

When `level>0`, G'MIC log messages are displayed on the standard error (stderr).

### Default values:

`level=1`.

---

# version

**No arguments**

## Description:

Display current version number on stdout.

---

# video2files

## Arguments:

- `input_filename,_output_filename,_first_frame>=0,_last_frame={ >=0 | -1=last },_frame_step>=1`

## Description:

Split specified input video file into image files, one for each frame.

First and last frames as well as step between frames can be specified.

## Default values:

`output_filename=frame.png`, `first_frame=0`, `last_frame=-1` and `frame_step=1`.

---

# vignette

## Arguments:

- `_strength>=0,0<=_radius_min<=100,0<=_radius_max<=100`

## Description:

Add vignette effect to selected images.

## Default values:

`strength=100`, `radius_min=70` and `radius_max=90`.

## Example of use:

```
image.jpg vignette ,
```

[0]: 'image.jpg' (640x427x1x3)

---

# volume3d

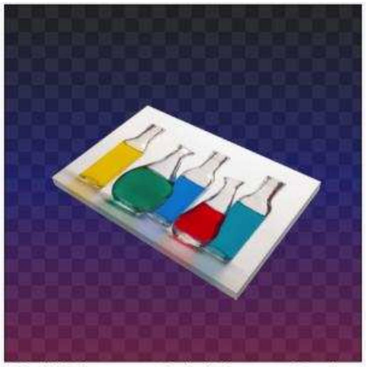
**No arguments**

## Description:

Transform selected 3D volumetric images as 3D parallelepipedic objects.

## Example of use:

```
image.jpg animate blur,0,5,30 append z volume3d
```



[0]: '[3D image cube]' (24 vert., 6 prim.)

---

# volumetric2d

## Arguments:

- `_x[%],_y[%],_z[%],_separator_size>=0`

## Description:

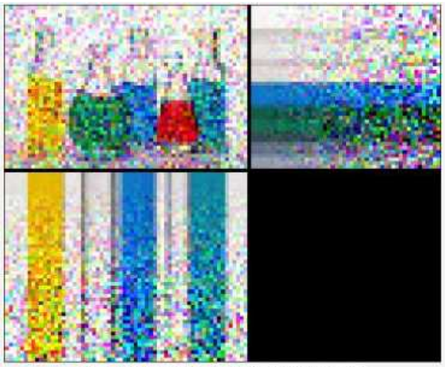
Convert selected 3D volumetric images into a 2D representation.

## Default values:

`x=y=z=50%` and `separator_size=0`.

### Example of use:

```
image.jpg rescale2d 64 animate noise,0,100,50 cut 0,255 append z
volumetric2d 50%,50%,50%,1
```



[0]: 'image_c1.jpg' (115x94x1x3)

## voronoi

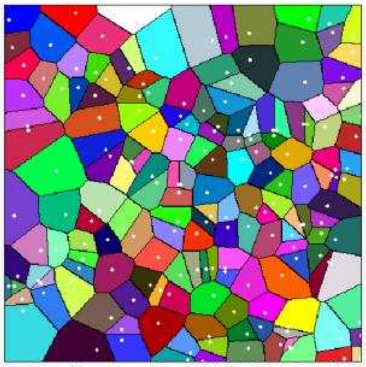
**No arguments**

### Description:

Compute the discrete Voronoi diagram of non-zero pixels in selected images.

### Example of use:

```
400,400 noise 0.2,2 eq 1 +label_fg 0 voronoi[-1] +gradient[-1] xy,1
append[-2,-1] c norm[-1] ==[-1] 0 map[-2] 2,2 mul[-2,-1]
normalize[-2] 0,255 dilate_circ[-2] 4 reverse max
```



[0]: '[unnamed]_c1' (400x400x1x3)

## voxelize3d

## Arguments:

- `_max_resolution>0,_fill_interior={ 0 | 1 },_preserve_colors={ 0 | 1 }`

## Description:

Convert selected 3D objects as 3D volumetric images of binary voxels, using 3D mesh rasterization.

## Default values:

`max_resolution=128`, `fill_interior=1` and `preserve_colors=0`.

---

# wait

Built-in command

## Arguments:

- `delay` or
- `(no arg)`

## Description:

Wait for a given delay (in ms), optionally since the last call to `wait`.

or wait for a user event occurring on the selected instant display windows.
`delay` can be ***{ <0:delay+flush events | 0:event | >0:delay }***.
Command selection (if any) stands for instant display window indices instead of image indices.
If no window indices are specified and if `delay` is positive, the command results
in a `hard` sleep during specified delay.

## Default values:

`delay=0`.

---

# warhol

## Arguments:

- `_M>0,_N>0,_smoothness>=0,_color>=0`

## Description:

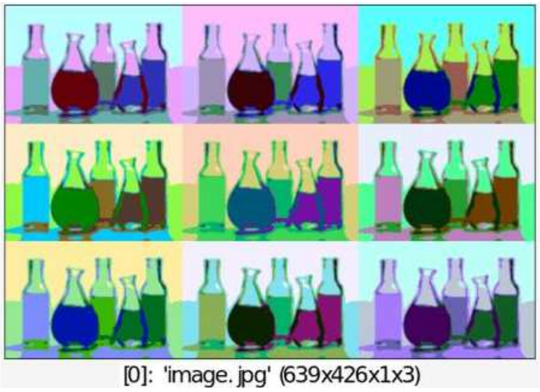
Create MxN Andy Warhol-like artwork from selected images.

## Default values:

`M=3`, `N=M`, `smoothness=2` and `color=20`.

## Example of use:

```
image.jpg warhol 3,3,3,40
```



[0]: 'image.jpg' (639x426x1x3)

## warn

Built-in command

### Arguments:

- `_force_visible={ 0 | 1 },_message`

### Description:

Print specified warning message, on the standard error (stderr).

Command selection (if any) stands for displayed call stack subset instead of image indices.

## warp

Built-in command

### Arguments:

- `[warping_field],_mode,_interpolation,_boundary_conditions,_nb_frames>0`

### Description:

Warp selected images with specified displacement field.

`mode` can be ***{ 0:backward-absolute | 1:backward-relative | 2:forward-absolute | 3:forward-relative }***.
`interpolation` can be ***{ 0:nearest-neighbor | 1:linear | 2:cubic }***.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

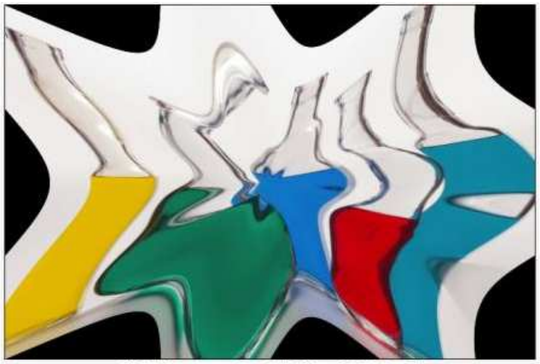
### Default values:

`mode=0`, `interpolation=1`, `boundary_conditions=0` and `nb_frames=1`.

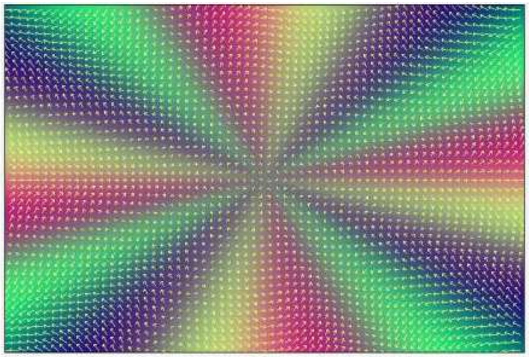
This command has a **tutorial page**.

### Example of use:

```
image.jpg 100%,100%,1,2,'X=x/w-0.5;Y=y/
h-0.5;R=(X*X+Y*Y)^0.5;A=atan2(Y,X);130*R*(!c?cos(4*A):sin(8*A))'
warp[-2] [-1],1,1,0 quiver[-1] [-1],10,1,1,1,100
```



[0]: 'image.jpg' (640x427x1x3)



[1]: '[X=x/w-0.5;Y=y/h-0.5;R=(X*X+Y*Y)^0.5;A=atan2(Y,X);130*R*(!c?cos(4*A):sin(8*A))]'
(640x427x1x2)

# warp_patch

### Arguments:

- `[warping_field],patch_width>=1,_patch_height>=1,_patch_depth>=1,_std_factor>0`

### Description:

Patch-warp selected images, with specified 2D or 3D displacement field (in backward-absolute mode).

Argument `std_factor` sets the std of the gaussian weights for the patch overlap,
equal to 'std = std_factor*patch_size'.
`boundary_conditions` can be ***{ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }***.

### Default values:

`std_factor=0.3` and `boundary_conditions=3`.

# warp_perspective

### Arguments:

- `_x-angle,_y-angle,_zoom>0,_x-center,_y-center,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

### Description:

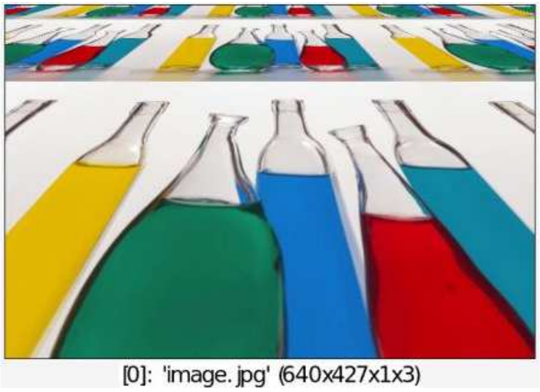
Warp selected images with perspective deformation.

## Default values:

`x-angle=1.5`, `y-angle=0`, `zoom=1`, `x-center=y-center=50` and `boundary_conditions=2`.

## Example of use:

```
image.jpg warp_perspective ,
```



[0]: 'image.jpg' (640x427x1x3)

# warp_rbf

## Arguments:

- `xs0[%],ys0[%],xt0[%],yt0[%],...,xsN[%],ysN[%],xtN[%],ytN[%]`

## Description:

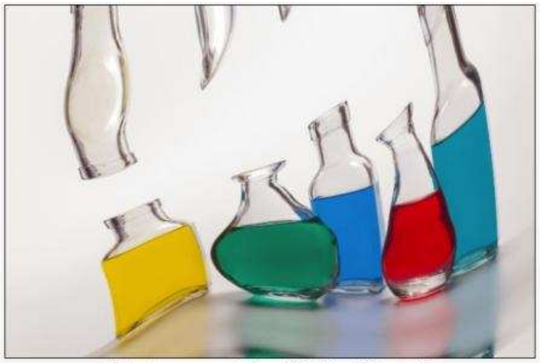
Warp selected images using RBF-based interpolation.

Each argument (xsk,ysk)-(xtk,ytk) corresponds to the coordinates of a keypoint respectively on the source and target images. The set of all keypoints define the overall image deformation.

## Example of use:

```
image.jpg +warp_rbf 0,0,0,0,100%,0,100%,0,100%,100%,100%,100%,0,100%,
0,100%,50%,50%,70%,50%,25%,25%,25%,75%
```

[0]: 'image.jpg' (640x427x1x3)


[1]: 'image_c1.jpg' (640x427x1x3)

---

# water

## Arguments:

- `_amplitude,_smoothness>=0,_angle`

## Description:

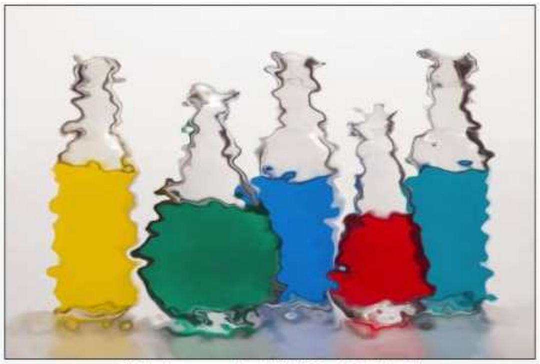
Apply water deformation on selected images.

## Default values:

`amplitude=30`, `smoothness=1.5` and `angle=45`.

## Example of use:

```
image.jpg water ,
```


[0]: 'image.jpg' (640x427x1x3)

---

# watermark_fourier

## Arguments:

- `text,_size>0`

### Description:

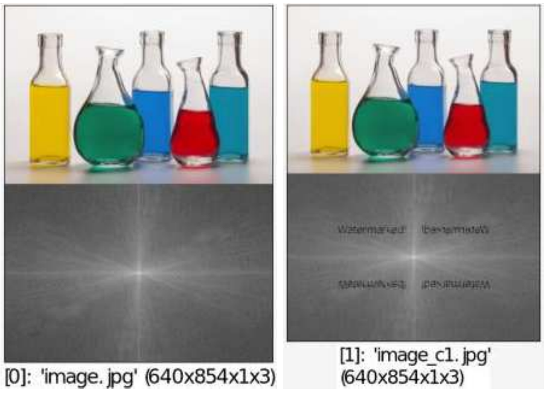
Add a textual watermark in the frequency domain of selected images.

### Default values:

`size=33`.

### Example of use:

```
image.jpg +watermark_fourier "Watermarked!" +display_fft remove[-3,
-1] normalize 0,255 append[-4,-2] y append[-2,-1] y
```



[0]: 'image.jpg' (640x854x1x3)

[1]: 'image_c1.jpg'
(640x854x1x3)

## watermark_visible

### Arguments:

- `_text,0<_opacity<1,_{ size>0 | font },_angle,_mode={ 0:remove | 1:add },_smoothness>=0`

### Description:

Add or remove a visible watermark on selected images (value range must be [0,255]).

### Default values:

'text=(c) G'MIC', `opacity=0.3`, `size=53`, `angle=25`, `mode=1` and `smoothness=0`.

### Example of use:

```
image.jpg watermark_visible ,0.7
```

[0]: 'image.jpg' (640x427x1x3)

## watershed

Built-in command

### Arguments:

- `[priority_image],_is_high_connectivity={ 0 | 1 }`

### Description:

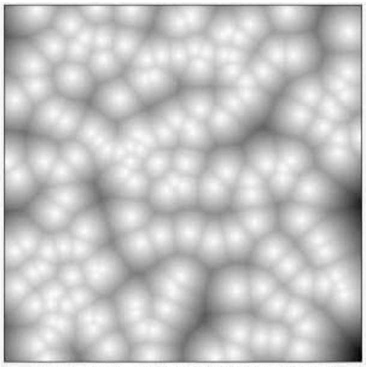
Compute the watershed transform of selected images.

### Default values:

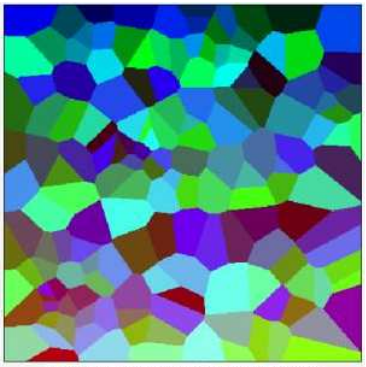
`is_high_connectivity=1`.

### Example of use:

```
400,400 noise 0.2,2 eq 1 +distance 1 mul[-1] -1 label[-2]
watershed[-2] [-1] mod[-2] 256 map[-2] 0 reverse
```



[0]: '[unnamed]_c1' (400x400x1x1)



[1]: '[unnamed]' (400x400x1x3)

## wave

### Arguments:

- `_amplitude>=0,_frequency>=0,_center_x,_center_y`

## Description:

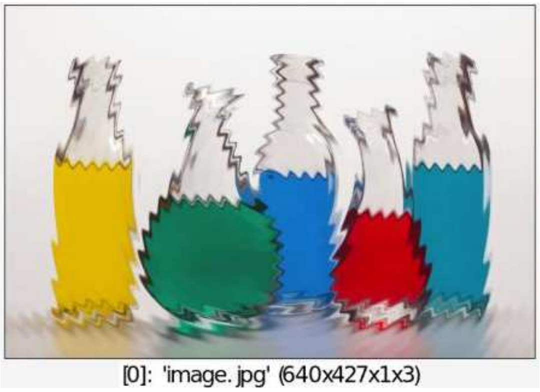
Apply wave deformation on selected images.

## Default values:

`amplitude=4`, `frequency=0.4` and `center_x=center_y=50`.

## Example of use:

```
image.jpg wave ,
```



[0]: 'image.jpg' (640x427x1x3)

---

# weave

## Arguments:

- `_density>=0,0<=_thickness<=100,0<=_shadow<=100,_shading>=0,_fibers_amplitude>` `-1<=_y_curvature<=1`

## Description:

Apply weave effect to the selected images.

`angle` can be ***{ 0:0 deg. | 1:22.5 deg. | 2:45 deg. | 3:67.5 deg. }***.

## Default values:

`density=6`, `thickness=65`, `shadow=40`, `shading=0.5`, `fibers_amplitude=0`, _'fibers_smoothness=0', `angle=0` and `curvature_x=curvature_y=0`

## Example of use:

```
image.jpg weave ,
```

[0]: 'image.jpg' (640x427x1x3)

# weird3d

## Arguments:

- `_resolution>0`

## Description:

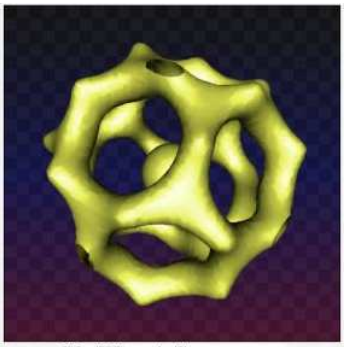
Input 3D weird object at (0,0,0), with specified resolution.

## Default values:

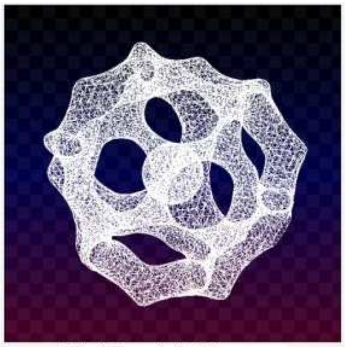
`resolution=32`.

## Example of use:

```
weird3d 48 +primitives3d 1 color3d[-2] ${-rgb}
```



[0]: '[3D weird]'
(11928 vert., 23784 prim.)



[1]: '[3D weird]_c1'
(11928 vert., 35736 prim.)

# while

Built-in command

## Arguments:

- `condition`

### Description:

End a `do...while` block and go back to associated `do` if specified condition holds.

`condition` is a mathematical expression, whose evaluation is interpreted as ***{ 0:false | other:true }***.

## whirls

### Arguments:

- `_texture>=0,_smoothness>=0,_darkness>=0,_lightness>=0`

### Description:

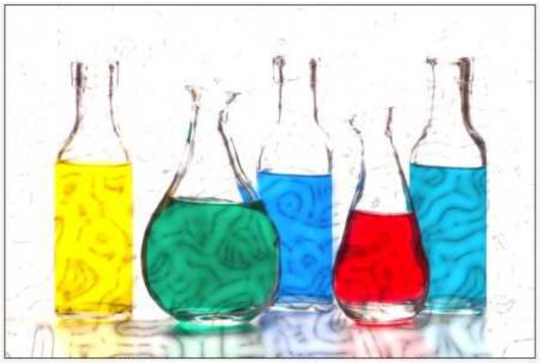
Add random whirl texture to selected images.

### Default values:

`texture=3`, `smoothness=6`, `darkness=0.5` and `lightness=1.8`.

### Example of use:

```
image.jpg whirls ,
```



[0]: 'image.jpg' (640x427x1x3)

## wind

### Arguments:

- `_amplitude>=0,_angle,0<=_attenuation<=1,_threshold`

### Description:

Apply wind effect on selected images.

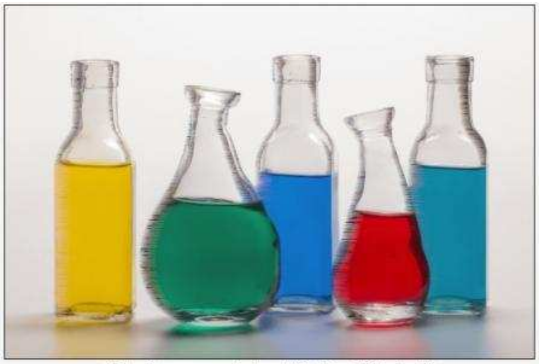
### Default values:

`amplitude=20`, `angle=0`, `attenuation=0.7` and `threshold=20`.

## Example of use:

```
image.jpg +wind ,
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

# window

**Built-in command**

## Arguments:

- `_width[%]>=-1,_height[%]>=-1,_normalization,_fullscreen,_pos_x[%],_pos_y[%],_`

## Description:

Display selected images into an instant display window with specified size, normalization type,

fullscreen mode and title.

(*equivalent to shortcut command* `w`).

If `width` or `height` is set to -1, the corresponding dimension is adjusted to the window or image size.
Specify `pos_x` and `pos_y` arguments only if the window has to be moved to the specified coordinates. Otherwise, they can be avoided.
'width'=0 or 'height'=0 closes the instant display window.
`normalization` can be ***`{ -1:keep same | 0:none | 1:always | 2:1st-time | 3:auto }`***.
`fullscreen` can be ***`{ -1:keep same | 0:no | 1:yes }`***.
You can manage up to 10 different instant display windows by using the numbered variants `w0` (default, eq. to `w`), `w1`,..., `w9` of the command `w`.
Invoke `window` with no selection to make the window visible, if it has been closed by the user.

## Default values:

`width=height=normalization=fullscreen=-1` and `title=(undefined)`.

# x_2048

**No arguments**

## Description:

Launch the 2048 game.

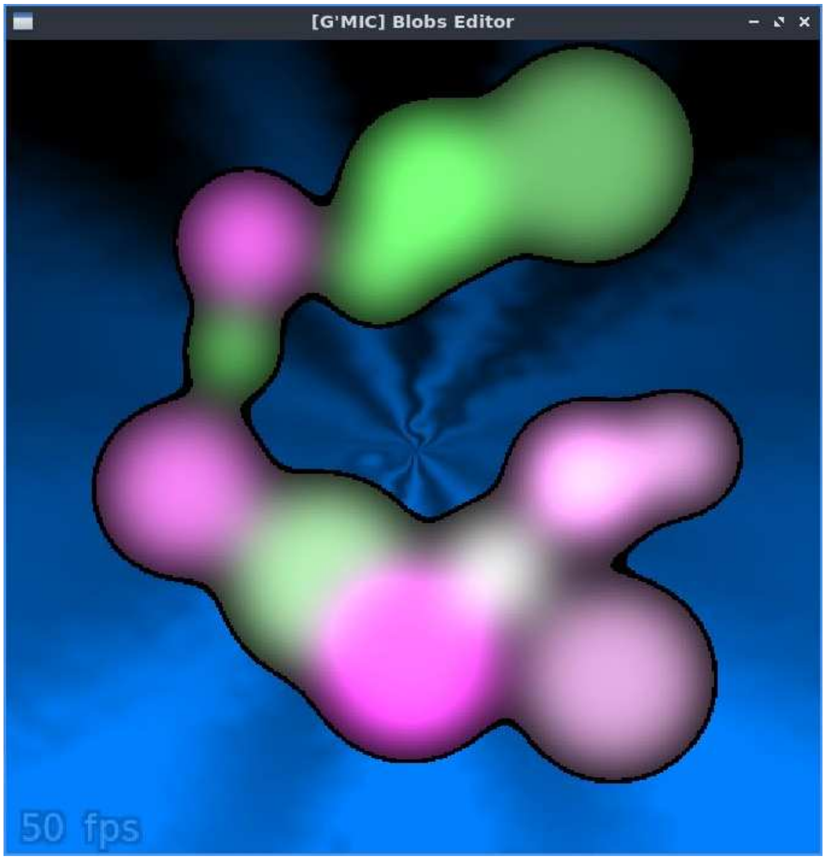
---

# x_blobs

**No arguments**

## Description:

Launch the blobs editor.



---

# x_bouncing

**No arguments**

## Description:

Launch the bouncing balls demo.

# x_color_curves

## Arguments:

- `_colorspace={ rgb | cmy | cmyk | hsi | hsl | hsv | lab | lch | ycbcr | last }`

## Description:

Apply color curves on selected RGB[A] images, using an interactive window.

Set `colorspace` to `last` to apply last defined color curves without opening interactive windows.

## Default values:

`colorspace=rgb`.

# x_colorize

## Arguments:

- `_is_lineart={ 0 | 1 },_max_resolution={ 0 | >=128 },_multichannels_output={ 0 | 1 },_[palette1],_[palette2],_[grabber1]`

## Description:

Colorized selected B&W images, using an interactive window.

When >0, argument `max_resolution` defines the maximal image resolution used in the interactive window.

## Default values:

`is_lineart=1`, `max_resolution=1024` and `multichannels_output=0`.

# x_connect4

### No arguments

## Description:

Launch the Connect Four game.

# x_crop

#### No arguments

## Description:

Crop selected images interactively.

If multiple input images are selected, the same crop is applied to all images.

(*equivalent to shortcut command* `xz`).

---

# x_cut

#### No arguments

## Description:

Cut selected images interactively.

---

# x_fire

#### No arguments

## Description:

Launch the fire effect demo.

---

# x_fireworks

#### No arguments

## Description:

Launch the fireworks demo.

---

# x_fisheye

#### No arguments

## Description:

Launch the fish-eye effect demo.

# x_fourier

**No arguments**

## Description:

Launch the fourier filtering demo.

# x_grab_color

## Arguments:

- `_variable_name`

## Description:

Open a color grabber widget from the first selected image.

Argument `variable_name` specifies the variable that contains the selected color values at any time.
Assigning `-1` to it forces the interactive window to close.

## Default values:

`variable_name=xgc_variable`.

# x_hanoi

**No arguments**

## Description:

Launch the Tower of Hanoi game.

# x_histogram

**No arguments**

## Description:

Launch the histogram demo.

## x_hough

**No arguments**

### Description:

Launch the hough transform demo.

## x_jawbreaker

### Arguments:

- `0<_width<20,0<_height<20,0<_balls<=8`

### Description:

Launch the Jawbreaker game.

## x_landscape

**No arguments**

### Description:

Launch the virtual landscape demo.

## x_life

**No arguments**

### Description:

Launch the game of life.

## x_light

**No arguments**

### Description:

Launch the light effect demo.

# x_mandelbrot

## Arguments:

- `_julia={ 0 | 1 },_c0r,_c0i`

## Description:

Launch Mandelbrot/Julia explorer.

# x_mask_color

## Arguments:

- `_colorspace={ all | rgb | lrgb | ycbcr | lab | lch | hsv | hsi | hsl | cmy | cmyk | yiq },_spatial_tolerance>=0,_color_tolerance>=0`

## Description:

Interactively select a color, and add an alpha channel containing the corresponding color mask.

Argument `colorspace` refers to the color metric used to compute color similarities, and can be basically
one of ***{ rgb | lrgb | ycbcr | lab | lch | hsv | hsi | hsl | cmy | cmyk | yiq }***.
You can also select one one particular channel of this colorspace, by setting `colorspace` as `colorspace_channel` (e.g. `hsv_h` for the hue).

## Default values:

`colorspace=all`, `spatial_tolerance=5` and `color_tolerance=5`.

# x_metaballs3d

### No arguments

## Description:

Launch the 3D metaballs demo.

# x_minesweeper

## Arguments:

- `8<=_width=<20,8<=_height<=20`

## Description:

Launch the Minesweeper game.

---

# x_minimal_path

### No arguments

## Description:

Launch the minimal path demo.

---

# x_morph

## Arguments:

- `_nb_frames>=2,_preview_fidelity={ 0:coarsest | 1:coarse | 2:normal | 3:fine | 4:finest }`

## Description:

Launch the interactive image morpher.

## Default values:

`nb_frames=16` and `preview_fidelity=3`.

---

# x_pacman

### No arguments

## Description:

Launch pacman game.

---

# x_paint

### No arguments

## Description:

Launch the interactive painter.

## x_plasma

**No arguments**

### Description:

Launch the plasma effect demo.

## x_quantize_rgb

### Arguments:

- `_nbcolors>=2`

### Description:

Launch the RGB color quantization demo.

## x_reflection3d

**No arguments**

### Description:

Launch the 3D reflection demo.

## x_rubber3d

**No arguments**

### Description:

Launch the 3D rubber object demo.

## x_segment

### Arguments:

- `_max_resolution={ 0 | >=128 }`

## Description:

Segment foreground from background in selected opaque RGB images, interactively.

Return RGBA images with binary alpha-channels.

## Default values:

`max_resolution=1024`.

---

# x_select_color

## Arguments:

- `_variable_name`

## Description:

Display a RGB or RGBA color selector.

Argument `variable_name` specifies the variable that contains the selected color values (as R,G,B,[A])
at any time.
Its value specifies the initial selected color. Assigning `-1` to it forces the interactive window to close.

## Default values:

`variable_name=xsc_variable`.

---

# x_select_function1d

## Arguments:

- `_variable_name,_background_curve_R,_background_curve_G,_background_curve_B`

## Description:

Open an interactive window, where the user can defined its own 1D function.

If an image is selected, it is used to display additional information :

  - The first row defines the values of a background curve displayed on the window (e.g. an histogram).
  - The 2nd, 3rd and 4th rows define the R,G,B color components displayed beside the X and Y axes.

Argument `variable_name` specifies the variable that contains the selected function keypoints at any time.
Assigning `-1` to it forces the interactive window to close.

## Default values:

`variable_name=xsf_variable`, `background_curve_R=220`, `background_curve_G=background_curve_B=background_curve_T`.

---

# x_select_palette

## Arguments:

- `_variable_name,_number_of_columns={ 0:auto | >0 }`

## Description:

Open a RGB or RGBA color selector widget from a palette.

The palette is given as a selected image.
Argument `variable_name` specifies the variable that contains the selected color values (as R,G,B,[A])
at any time.
Assigning `-1` to it forces the interactive window to close.

## Default values:

`variable_name=xsp_variable` and `number_of_columns=2`.

---

# x_shadebobs

### No arguments

## Description:

Launch the shade bobs demo.

---

# x_spline

### No arguments

## Description:

Launch spline curve editor.

---

# x_starfield3d

**No arguments**

## Description:

Launch the 3D starfield demo.

---

# x_tetris

**No arguments**

## Description:

Launch tetris game.

---

# x_threshold

**No arguments**

## Description:

Threshold selected images interactively.

---

# x_tictactoe

**No arguments**

## Description:

Launch tic-tac-toe game.

---

# x_warp

## Arguments:

- `_nb_keypoints_xgrid>=2,_nb_keypoints_ygrid>=2,_nb_keypoints_contours>=0,_prev`
  `0:coarsest | 1:coarse | 2:normal | 3:fine | 4:finest`
  `},_[background_image],0<=_background_opacity<=1`

## Description:

Launch the interactive image warper.

## Default values:

`nb_keypoints_xgrid=nb_keypoints_ygrid=2`, `nb_keypoints_contours=0` and `preview_fidelity=1`.

---

# x_waves

**No arguments**

## Description:

Launch the image waves demo.

---

# x_whirl

## Arguments:

- `_opacity>=0`

## Description:

Launch the fractal whirls demo.

## Default values:

`opacity=0.2`.

---

# xor

Built-in command

## Arguments:

- `value[%]` or
- `[image]` or
- `'formula'` or
- `(no arg)`

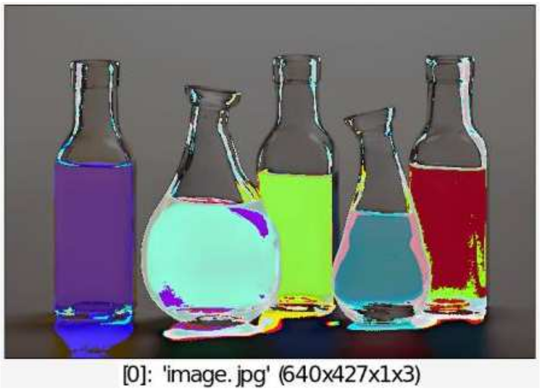
## Description:

Compute the bitwise XOR of selected images with specified value, image or mathematical expression, or compute the pointwise sequential bitwise XOR of selected images.

## Examples of use:

**• Example #1**

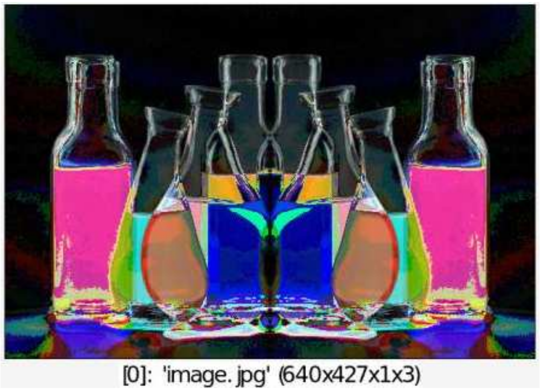
```
image.jpg xor 128
```



[0]: 'image.jpg' (640x427x1x3)

- **Example #2**

```
image.jpg +mirror x xor
```



[0]: 'image.jpg' (640x427x1x3)

---

## xyz2jzazbz

**No arguments**

### Description:

Convert color representation of selected images from XYZ to RGB.

---

## xyz2lab

### Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

### Description:

Convert color representation of selected images from XYZ to Lab.

## Default values:

`illuminant=2`.

---

# xyz2rgb

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from XYZ to RGB.

## Default values:

`illuminant=2`.

---

# xyz82rgb

## Arguments:

- `illuminant={ 0:D50 | 1:D65 | 2:E }` or
- `(no arg)`

## Description:

Convert color representation of selected images from XYZ8 to RGB.

## Default values:

`illuminant=2`.

---

# ycbcr2rgb

### No arguments

## Description:

Convert color representation of selected images from YCbCr to RGB.

---

## yinyang

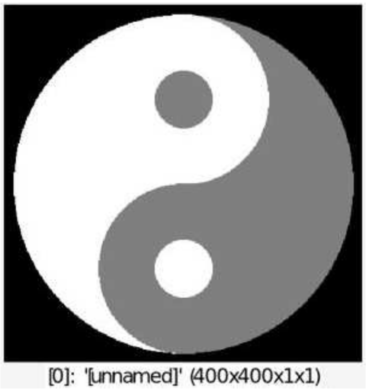
**No arguments**

### Description:

Draw a yin-yang symbol on selected images.

### Example of use:

```
400,400 yinyang
```



---

## yiq2rgb

**No arguments**

### Description:

Convert color representation of selected images from YIQ to RGB.

---

## yiq82rgb

**No arguments**

### Description:

Convert color representation of selected images from YIQ8 to RGB.

---

## yuv2rgb

**No arguments**

### Description:

Convert color representation of selected images from YUV to RGB.

---

# yuv82rgb

### No arguments

## Description:

Convert selected images from YUV8 to RGB color bases.

---

# zoom

## Arguments:

- `_factor,_cx,_cy,_cz,_boundary_conditions={ 0:dirichlet | 1:neumann | 2:periodic | 3:mirror }`

## Description:

Apply zoom factor to selected images.

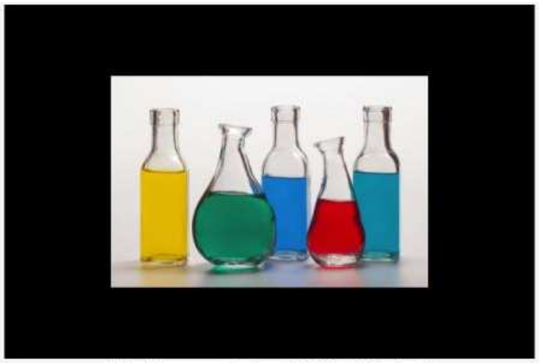
## Default values:

`factor=1`, `cx=cy=cz=0.5` and `boundary_conditions=0`.

## Example of use:

```
image.jpg +zoom[0] 0.6 +zoom[0] 1.5
```

[0]: 'image.jpg' (640x427x1x3)



[1]: 'image_c1.jpg' (640x427x1x3)

[2]: 'image_c1. jpg' (640x427x1x3)

□ End of document