SUCKLESS IMAGE PROCESSING

Enric Meinhardt-Llopis

SLCON3 -- Taunus -- 24/9/2016

Contents

Image Processing in UNIX

* pbmutils

```
cat lena.png | pngtopnm | pnmsmooth -size 7 7 | pnmtopng > out.png
```

* Imagemagick

```
cat lena.png | convert - -gaussian-blur 7x7+0+0 out.png
```

* GMIC

```
cat lena.png | gmic -blur 7  -output -.png > out.png
```

* imscript (our proposal)

```
cat lena.png | blur gaussian 7 - out.png
```

Image Processing in UNIX

* pbmutils

```
cat lena.png | pngtopnm | pnmsmooth -size 7 7 | pnmtopng > out.png
```

* Imagemagick

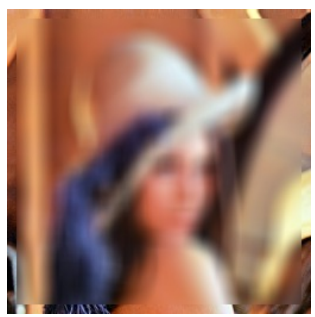
```
cat lena.png | convert - -gaussian-blur 7x7+0+0 out.png
```

* GMIC

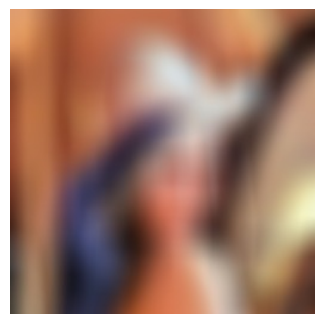
```
cat lena.png | gmic -blur 7  -output -.png > out.png
```

* imscript (our proposal)

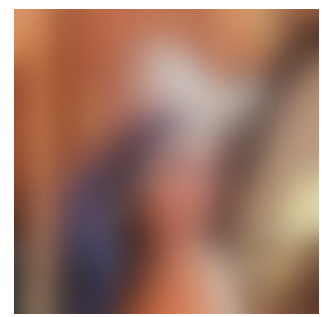
```
cat lena.png | blur gaussian 7 - out.png
```



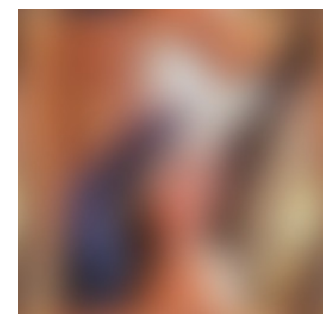
pnm



magick



gmic



imscript

GMIC is the choice that sucks less

- v based on cimg.h, a beautiful single-file library
- v appropriate for general image processing
- v much better API than imagemagick
- v elegant and generic "asc" interchange format
- x written in C++
- v thin wrapper for cimg.h
- x single, monolithic executable
- v single, monolithic executable
- x very slow compilation (huge, templated C++)

why imscript?

- v written in C
- v based on the elegant NIPL library
- v each task done by an independent program

why imscript?

- v written in C
- v based on the elegant NIPL library
- v each task done by an independent program
- v incomplete
- v inconsistent
- v many ad-hoc, generally useless programs (329)
- v ugly makefile
- v ugly names

imscript tools

```
plambda:  apply an expression with images as variables
blur:     convolution by a positive kernel
downsa:   zoom out
upsa:     zoom in
ntiply:   replicate pixels
crop:     extract a sub-image
fft/iift: discrete Fourier transform of an image, and inverse
dct,dht:  discrete cosine,Hartley transforms (self-inverse)
imprintf: print info and statistics of an image
vecov:    accumulate values over a list of images
morsi:    elementary gray-level morphological operations
simpois:  poisson and bilaplacian iterative solvers
homwarp:  deform an image by a homographic map
fontu:    overlay a text in an image
pview:    draw points and lines specified by stdin text
iion:     copy (useful for converting between formats)
```

```
PLAMBDA(1)

NAME
     plambda - evaluate an expression with images as variables

SYNOPSIS
     plambda a.png b.png ... "EXPRESSION" > output
     plambda a.png b.png ... "EXPRESSION" -o output

DESCRIPTION
     Plambda evaluates an expression with images as variables.

     The expression is written in reverse polish notation using
     arithmetic operators and functions from ‘math.h’.

EXAMPLES
     plambda a.tiff b.tiff + > sum.tiff
          Compute the sum of two images
```

lena.png

peppers.png

plambda lena.png peppers.png + | d

```
PLAMBDA(1)

NAME
     plambda - evaluate an expression with images as variables

SYNOPSIS
     plambda a.png b.png ... "EXPRESSION" > output
     plambda a.png b.png ... "EXPRESSION" -o output

DESCRIPTION
     Plambda evaluates an expression with images as variables.

     The expression is written in reverse polish notation using
     arithmetic operators and functions from ‘math.h’.

EXAMPLES
     plambda a.tiff b.tiff + > sum.tiff
          Compute the sum of two images
```

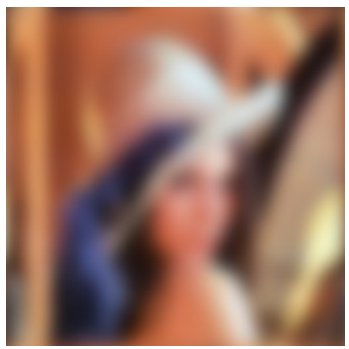
lena.png

peppers.png

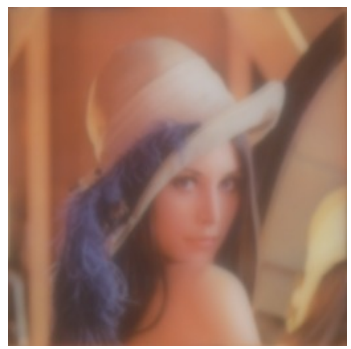
plambda lena.png peppers.png "+ 2 /" | d

```
BLUR(1)

NAME
     blur - image convolution by a positive kernel

SYNOPSIS
     blur kernel_name radius [in [out]]

DESCRIPTION
     Blur computes the convolution of an image by the
     typical positive kernels: gaussian, laplace, cauchy, disk...

EXAMPLES
     plambda laplace 0.6 lena.png dequantized_lena.png
     plambda gauss   1.6 lena.png blurred_lena.png
```

lena.png



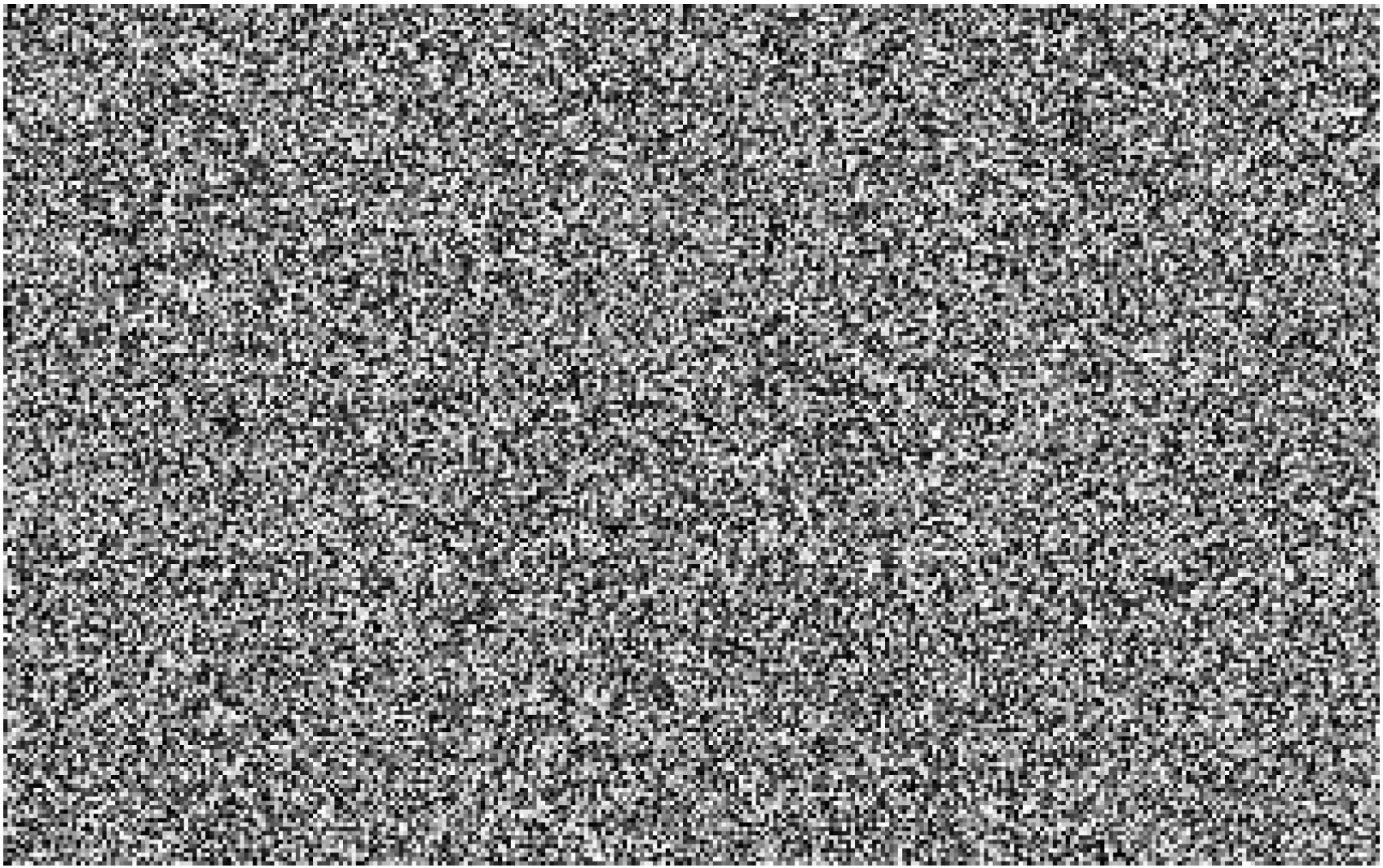
blur gauss 7



blur cauchy 1

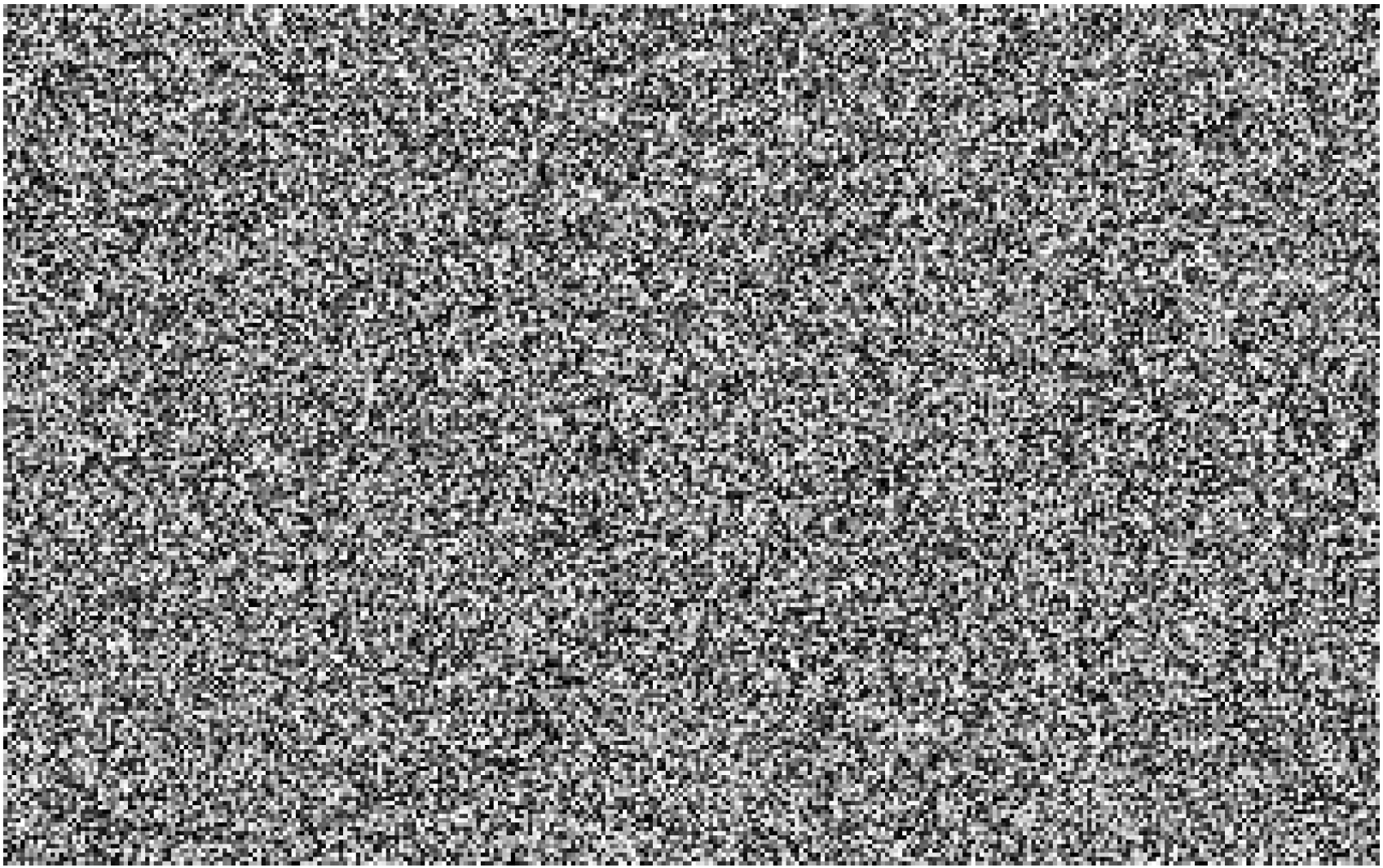
```
IMPRINTF(1)

NAME
     imprintf - print info and statistics of an image to stdout

SYNOPSIS
     imprintf "format" image

DESCRIPTION
     This program displays image information and statistics
     specified using a printf(1)-like string.  The following
     conversions are recognized:
          w    image width
          h    image height
          c    pixel dimension
          n,N  number of samples,pixels
          v,V  average sample,pixel
          i,I  minimum sample,pixel
          a,A  maximum sample,pixel
          m,M  median sample,pixel
          r    root mean square of all samples
          e    average absolute value of all samples
EXAMPLES
      $ imprintf "size=%wx%d, values between %i and %a (average=%v)\n" lena.png
      size=256x256, values between 0 and 255 (average=119.079)
```

imscript usage examples



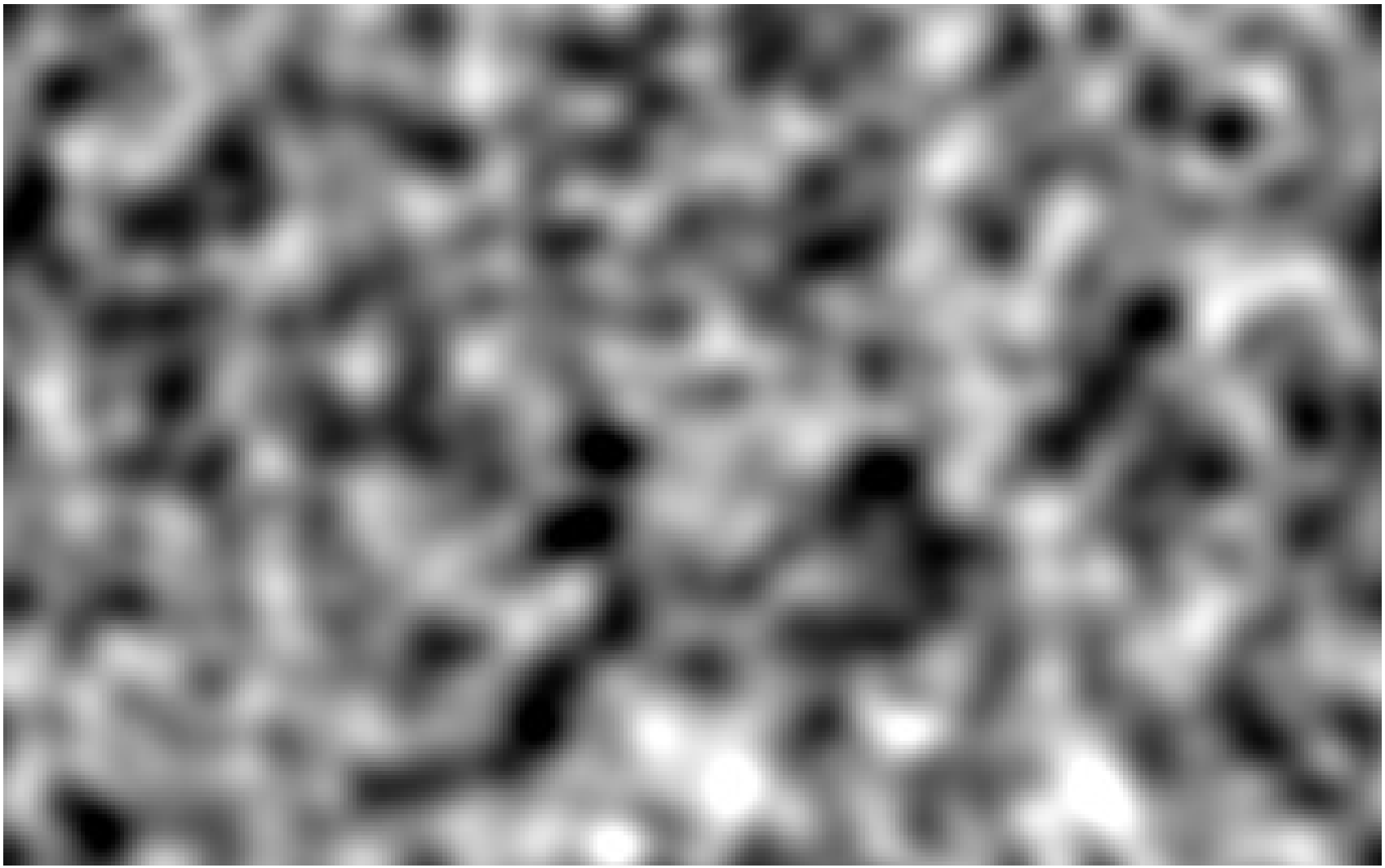
```
plambda zero:320x200 "rand fabs 255 fmod" | d
```

imscript usage examples



```
plambda zero:320x200 rand | qauto| d
```

imscript usage examples



```
plambda zero:320x200 rand | blur gaussian 5 | qauto| d
```

imscript usage examples


```
plambda zero:320x200 randp | blur cauchy 1 | qauto | fontu puts 20 70 -c 070 "hello world" | d
```

serious imscript example: a script for removing reflections

i00.jpg

serious imscript example: a script for removing reflections

i01.jpg

serious imscript example: a script for removing reflections

i02.jpg

serious imscript example: a script for removing reflections

i03.jpg

serious imscript example: a script for removing reflections

i04.jpg

serious imscript example: a script for removing reflections

i05.jpg

serious imscript example: a script for removing reflections

i06.jpg

serious imscript example: a script for removing reflections

i07.jpg

serious imscript example: a script for removing reflections

i08.jpg

serious imscript example: a script for removing reflections

i09.jpg

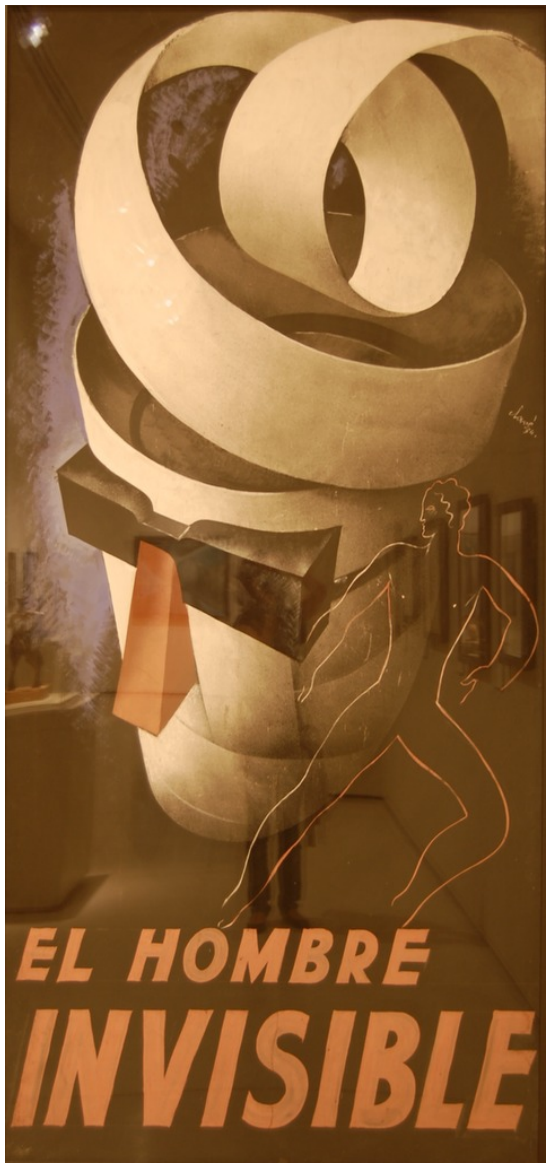
serious imscript example: a script for removing reflections

i10.jpg

serious imscript example: a script for removing reflections

i11.jpg

serious imscript example: a script for removing reflections

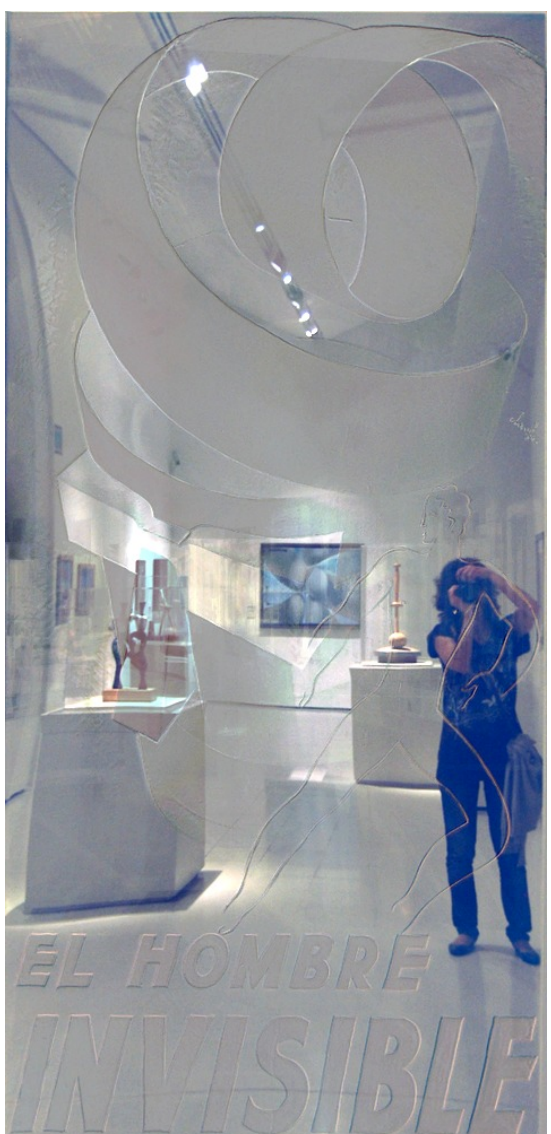


i00.png

out_med.png

serious imscript example: a script for removing reflections

reg_i05.png



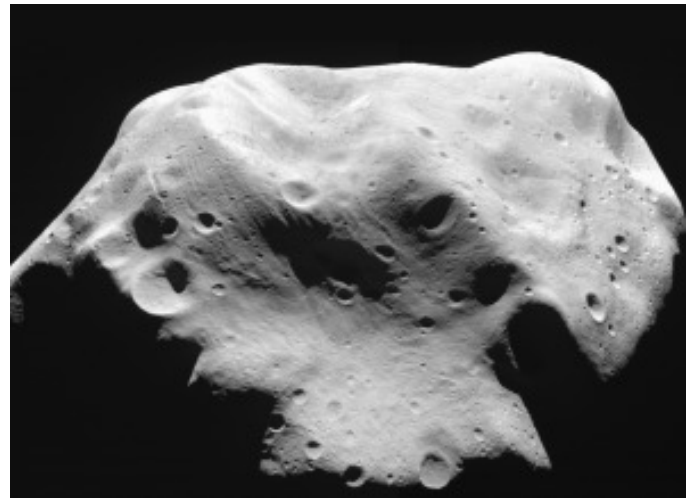

```
plambda reg_i05.png out_med.png - | qauto | c
```

serious imscript example: a script for removing reflections

```
IDX=‘seq -w 0 11‘
SIZE=‘imprintf "%w %h" i00.png‘

# compute sift descriptors of each image
for i in $IDX; do
        sift i$i.png > i$i.sift
done

# register each image to the first one
for i in $IDX; do
        siftu pairR 0.8 i00.sift i$i.sift p$i.txt  # match pairs
        ransac hom 1000 1 30 h$i.txt < p$i.txt     # find homography
        homwarp h$i.txt $SIZE i$i.png reg_$i.png   # warp
done

# compute the median value at each position
vecov med reg_*.png -o out_med.png
```

another serious example: removal of periodic noise

in.png



cat in.png | fft | plambda - m.png * | ifft | d

Requirements for an image format passed through pipes

* allow negative numbers
* allow floating-point values
* arbitrary number of channels

Observation: the color channels (or "bands")
must not have meaning. Do not think about RGB or RGBA
images, think about a 3 our a 4 channel image.

Motto: AN IMAGE IS AN ARRAY OF NUMBERS
(not colors, not intensities, not luminances)

Motto 2: AN IMAGE IS NOT A PHOTO
(e.g., astronomy, microscopy, geography...)

Image processing in C

Completely independent components:

* NIPL image processing library
* iio image input/output
* fontu font rendering
* siftu keypoint matching
* ftr window/mouse/keyboard interaction
* fancy_image deal transparently with huge images

NIPL: not an image processing library

NIPL: not an image processing library

A set of conventions for image processing in C

NIPL: not an image processing library

A set of conventions for image processing in C

1. Conventions for image data:
    1.1. An image is an array of pixels
    1.2. The pixels are ordered from left to right, and from top to bottom
    1.3. A pixel is an array of samples
    1.4. A sample is a numeric C type (typically uint8_t or float)
    1.5. All the pixels of an image have the same number of samples
    1.6. This number of samples is arbitrary, it can be 1, 3, or 10 or 1000
    1.7. The samples are numbers, not intensities
    1.8. Pixels are contiguous, never broken (synonymous to 1.3)
    1.9. Separate color planes, when needed, are just arrays of images
    1.10. The "default" format for samples is "float"

2. Coding conventions
    2.1. Structs are never used in the interface of a reusable algorithm
    2.2. Structs are OK for internal implementation details
    2.3. In particular, THERE IS NO USER-VISIBLE IMAGE STRUCT
    2.4. Algorithms may use an internal image struct adapted to their needs
    2.5. Typical names for image arguments: x, y, z, in, out
    2.6. Names for image sizes: w (width), h (height), pd (pixel dimension)
    2.7. Image sizes as arguments go together with their images
    2.8. Example of 2.7:
        void zoom_out(float *y, int wy, int hy, float *x, int wx, int hx)
    2.9. How to traverse all the pixels of an image (note the indentation):
```
for (int j = 0; j < h; j++)
for (int i = 0; i < w; i++)
        x[j*w + i] = 0;
```
    2.10. Order of function arguments: (output, input, options)
    2.11. Exception to 2.10: (state, output, input, options)

IIO: image input/output

IIO reads and writes numbers between image files and C arrays.
The filename "-" stands for stdin/stdout.
Extensions are ignored when reading, used when writing.

```c
void darken_image_inplace(float *x, int w, int h)
{
        for (int i = 0; i < w * h; i++)
                x[i] = x[i] / 2;
}

#include "iio.h"
int main()
{
        int w, h;
        float *x = iio_read_float("image.png", &w, &h);

        darken_image_inplace(x, w, h);

        iio_save_float("image_dark.png", x, w, h);
free(x);
        return 0;
}
```

FTR: suckless user-interface library

Minimal FTR program (to compile: cc mini.c -lx11)

```c
#include "ftr.c"

int main(void)
{
	struct FTR f = ftr_new_window(320, 200);
	return ftr_loop_run(&f);
}
```

Program that draws a green pixel on a black window

```c
#include "ftr.c"

int main(void)
{
	struct FTR f = ftr_new_window(320, 200);
	f.rgb[1 + 3 * (f.w * 10 + 10)] = 255;
	f.changed = 1;
	return ftr_loop_run(&f);
}
```

Interactive crop: cat lena.png | icrop > lena_part.png

Image tee: cat lena.png | itee | ...

FTR: suckless user-interface library

```
#ifndef _FTR_H
#define _FTR_H

struct FTR {
        int w, h;           // visible state
        unsigned char *rgb;
        int changed;
        void *userdata;     // ignored by the library
        char pad[200];      // implementation details
};

// type of a handler function
typedef void (*ftr_event_handler_t)(struct FTR*,int,int,int,int);

// window management
struct FTR ftr_new_window(int w, int h);
void ftr_change_title(struct FTR *f, char *title);
void ftr_close(struct FTR *f);

// event loop
int ftr_loop_run(struct FTR *f); // returns when the loop is finished
void ftr_notify_the_desire_to_stop_this_loop(struct FTR *f, int return_value);
int ftr_set_handler(struct FTR *f, char *id, ftr_event_handler_t e);

// key modifiers (numbers inspired by X)
#define FTR_MASK_SHIFT     1
...
// button modifiers (inspired by X)
#define FTR_BUTTON_LEFT    256
...
#endif//_FTR_H
```

END
===

Conclusions:
---

* UNIX is alright for image processing
  (no need for matlab, python+numpy and the like)

* C is alright for image processing
  (no need for specific image processing libraries)

* We have a need for a *general* image format

Interactive experience:
---

* Open a terminal and show things

Links

* http://www.ipol.im
IPOL: a journal of image processing algorithms

* https://tools.ipol.im/wiki/ref/software_guidelines/
The suckless IPOL software guidelines

* https://github.com/mnhrdt/imscript
My collection of unix image processing tools