NAME

plambda − evaluate an expression with images as variables

SYNOPSIS

plambda *img1.png img2.png img3.png ... "EXPRESSION"*
plambda *-c num1 num2 num3 ... "EXPRESSION"*

DESCRIPTION

Plambda evaluates an expression with images as variables.

The resulting image is printed to standard output. The expression should be written in reverse polish notation using common operators and functions from ‘math.h’. The variables appearing on the expression are assigned to each input image in alphabetical order.

EXPRESSIONS:

A "plambda" expression is a sequence of tokens. Tokens may be constants, variables, or operators. Constants and variables get their value computed and pushed to the stack. Operators pop values from the stack, apply a function to them, and push back the results.

CONSTANTS: numeric constants written in scientific notation, and "pi"

OPERATORS: +, −, *, ˆ, /, <, >, ==, and all the functions from math.h

LOGIC OPS: if, and, or, not

VARIABLES: anything not recognized as a constant or operator. There must be as many variables as input images, and they are assigned to images in alphabetical order. If there are no variables, the input images are pushed to the stack.

All operators (unary, binary and ternary) are vectorizable. Thus, you can add a scalar to a vector, divide two vectors of the same size, and so on. The semantics of each operation follows the principle of least surprise.

Some "sugar" is added to the language:

Predefined variables (always preceeded by a colon):

:i	horizontal coordinate of the pixel
:j	vertical coordinate of the pixel
:w	width of the image
:h	heigth of the image
:n	number of pixels in the image
:x	relative horizontal coordinate of the pixel
:y	relative horizontal coordinate of the pixel
:r	relative distance to the center of the image
:t	relative angle from the center of the image
:I	horizontal coordinate of the pixel (centered)
:J	vertical coordinate of the pixel (centered)
:W	width of the image divided by 2*pi
:H	height of the image divided by 2*pi

Variable modifiers acting on regular variables:

x	value of pixel (i,j)
x(0,0)	value of pixel (i,j)
x(1,0)	value of pixel (i+1,j)

x(0,−1)	value of pixel (i,j−1)
x[0]	value of first component of pixel (i,j)
x[1]	value of second component of pixel (i,j)
x(1,2)[3]	value of fourth component of pixel (i+1,j+2)

Stack operators (allow direct manipulation of the stack):

del	remove the value at the top of the stack (ATTTOS)
dup	duplicate the value ATTTOS
rot	swap the two values ATTTOS
split	split the vector ATTTOS into scalar components
join	join the components of two vectors ATTOTS
join3	join the components of three vectors ATTOTS
njoin	join the components of n vectors
halve	split an even−sized vector ATTOTS into two equal−sized parts

Magic variable modifiers (global data associated to each input image):

x%i	value of the smallest sample of image x
x%a	value of the _largest_ sample
x%v	average sample value
x%m	median sample value
x%s	sum of all samples
x%I	value of the smallest pixel (in euclidean norm)
x%A	value of the largest pixel
x%V	average pixel value
x%S	sum of all pixels
x%Y	component−wise minimum of all pixels
x%E	component−wise maximum of all pixels
x%qn	nth sample percentile
x%On	component−wise nth percentile
x%Wn	component−wise nth millionth part
x%0n	component−wise nth order statistic
x%9n	component−wise nth order statistic (from the right)

Random numbers:

randu	push a random number with distribution Uniform(0,1)
randn	push a random number with distribution Normal(0,1)
randc	push a random number with distribution Cauchy(0,1)
randl	push a random number with distribution Laplace(0,1)
rande	push a random number with distribution Exponential(1)
randp	push a random number with distribution Pareto(1)
rand	push a random integer returned from rand(3)

Vectorial operations (acting over vectors of a certain length):

topolar	convert a 2−vector from cartesian to polar
frompolar	convert a 2−vector from polar to cartesian
hsv2rgb	convert a 3−vector from HSV to RGB
rgb2hsv	convert a 3−vector from RGB to HSV
cprod	multiply two 2−vectrs as complex numbers
mprod	multiply two 2−vectrs as matrices (4−vector = 2x2 matrix, etc)
vprod	vector product of two 3−vectors
sprod	scalar product of two n−vectors
mdet	determinant of a n−matrix (a n*n−vector)
mtrans	transpose of a matrix
mtrace	trace of a matrix
minv	inverse of a matrix

Registers (numbered from 1 to 9):

>7	copy to register 7
<3	copy from register 3

OPTIONS

−c	act as a symbolic calculator
−h	display short help message
−−help	display longer help message

EXAMPLES

```
plambda a.tiff b.tiff "x y +" > sum.tiff
```

Compute the sum of two images.

```
plambda −c "1 atan 4 *"
```

Print pi

```
plambda −c "355 113 /"
```

Print an approximation of pi

AUTHOR

Written by Enric Meinhardt−Llopis

REPORTING BUGS

Report bugs to <enric.meinhardt@cmla.ens−cachan.fr>.