Real-Time Computer Vision with Ruby and LibJIT

J. Wedekind, M. Howarth, J. Travis, K. Dutton, B. Amavasai
Microsystems & Machine Vision Laboratory, Sheffield Hallam University, Sheffield, United Kingdom
EPSRC GR/S85696/01, Basic Technology: Nanorobotics
Technologies for Simultaneous Multidimensional Imaging & Manipulation of Nanoobjects

Abstract

Many image processing algorithms can be broken down into atomic operations on arrays. There are unary operations such as additive inverse, absolute value, or square-root and binary operations such as addition or multiplication. While the arrays typically have elements of a single type, this element-type may be a signed or unsigned integer with 8, 16, 32, or more bits, a floating point number, or a complex number.

A computer vision library thus needs to implement binary operations for various combinations of element-types. Furthermore binary operations can occur as array-array-, array-scalar-, or as scalar-array-operation. This kind of requirements ake it very hard to write a computer vision library in a statically typed language such as C++. On the other hand a naive implementation in a dynamically typed programming language such as Ruby will not satisfy real-time constraints.

However recently the DotGNU project has released libJIT which is a just-in-time compiler library for i386, x86-64, and other processors. The combination of Ruby, libJIT, and other free software allows interactive development of real-time algorithms in an unprecedented way.

Previous work: [WAD07], [WADB08], [HOR]

References

[HOR] Hornetseye. Web page. http://hornetseye.rubyforge.org/.

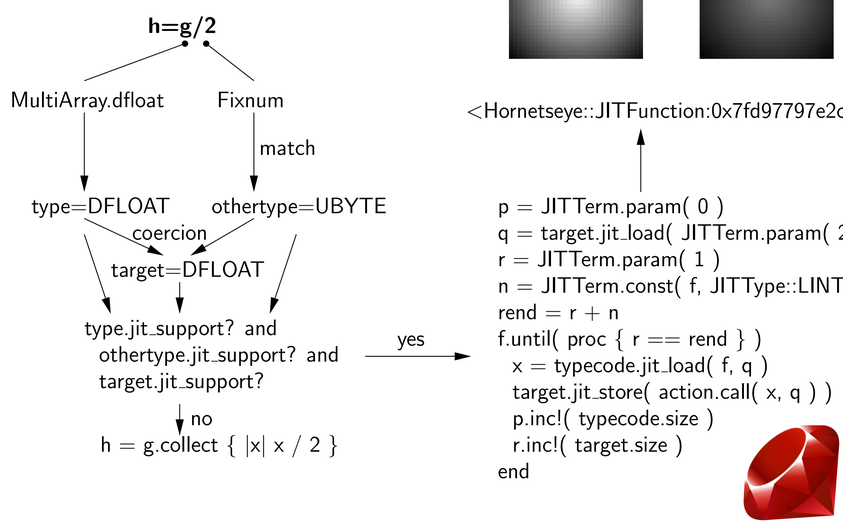
[WAD07] J. Wedekind, B. Amavasai, and K. Dutton. Steerable filters generated with the hypercomplex dual-tree wavelet transform. In *IEEE International Conference on Signal Processing and Communications*, pages 1291–4, 2007.

[WADB08] J. Wedekind, B. Amavasai, K. Dutton, and M. Boissenin. A machine vision extension for the Ruby programming language. pages 991–6, 2008.

HornetsEye, libJIT, Ruby

$$g,h \in \{0,1,\ldots,w-1\} \times \{0,1,\ldots,h-1\}$$

$$h(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}) = g(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix})/2$$

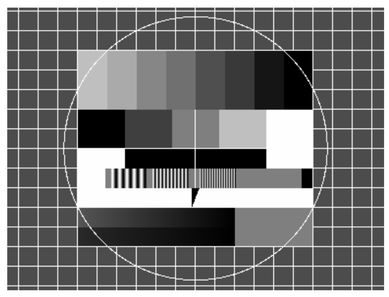


Shi-Tomasi Corner Detector

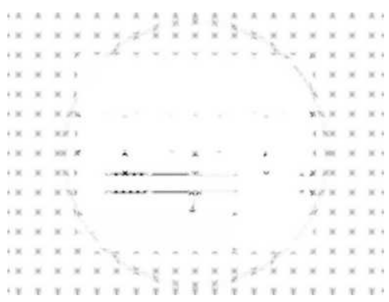
$$\mathcal{C} := \iint_W \begin{pmatrix} \left(\frac{\delta g(\vec{x})}{\delta x_1}\right)^2 & \frac{\delta g(\vec{x})}{\delta x_1} \cdot \frac{\delta g(\vec{x})}{\delta x_2} \\ \frac{\delta g(\vec{x})}{\delta x_1} \cdot \frac{\delta g(\vec{x})}{\delta x_2} & \left(\frac{\delta g(\vec{x})}{\delta x_2}\right)^2 \end{pmatrix} d\vec{x}$$

$$\mathcal{C} = \mathcal{A}^\top \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \mathcal{A}, \text{ Shi-Tomasi: } \min(\lambda_1,\lambda_2) > \lambda$$

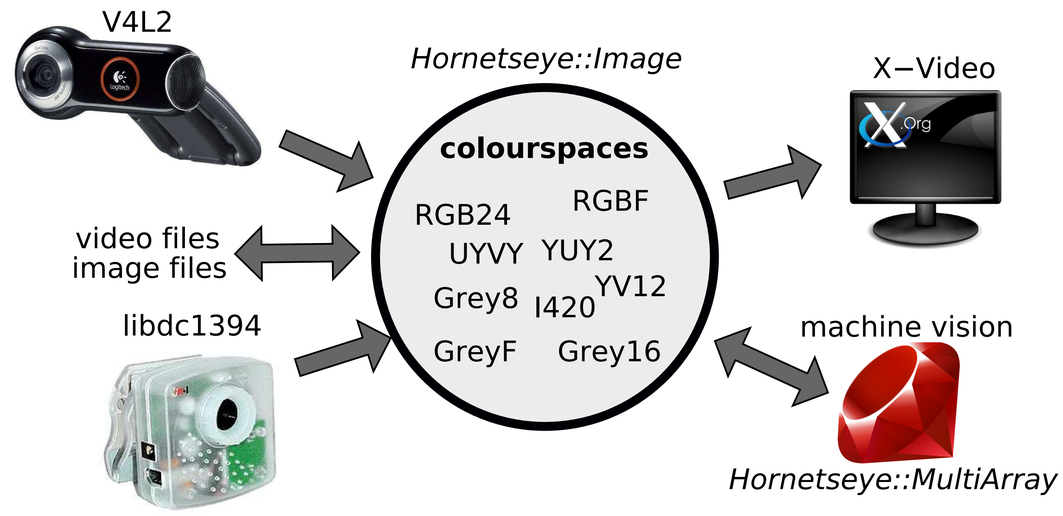
```
require 'hornetseye'
include Hornetseye
grad_sigma, cov_sigma = 1.0, 1.0
img = MultiArray.load_grey8 'test.png'
x = img.gauss_gradient_x grad_sigma
y = img.gauss_gradient_y grad_sigma
a = ( x ** 2 ).gauss_blur cov_sigma
b = ( y ** 2 ).gauss_blur cov_sigma
c = ( x * y ).gauss_blur cov_sigma
tr = a + b
det = a * b - c * c
dissqrt = Math.sqrt( ( tr * tr - det * 4 ).
                     major( 0.0 ) )
result = 0.5 * ( tr - dissqrt )
result.normalise( 255..0 ).show
```



⇓



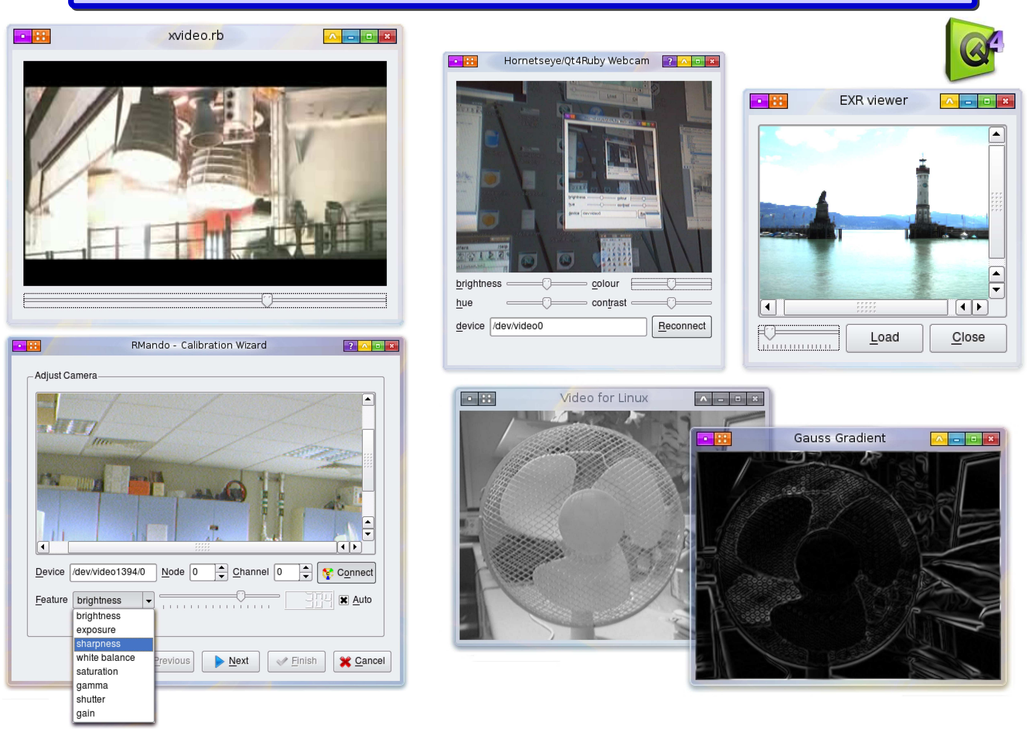
Colourspaces & I/O



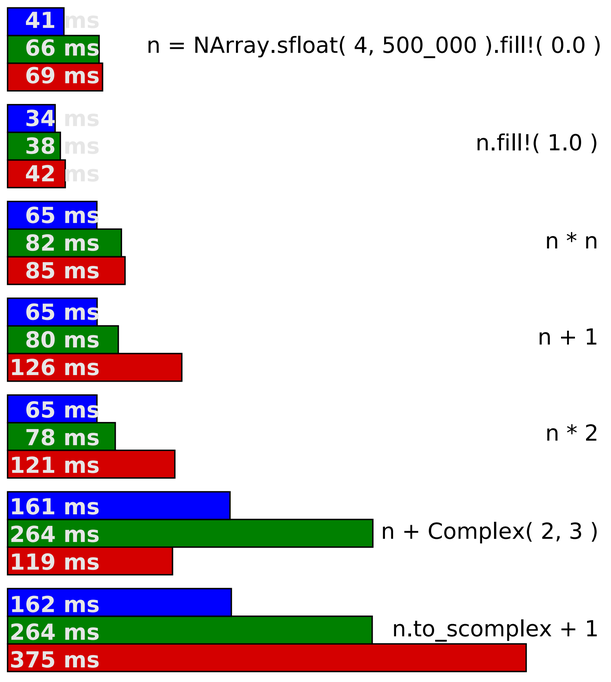
$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.500 \\ 0.500 & -0.418688 & -0.081312 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

also see: http://fourcc.org/

GUI integration



Computing Time



Comparison

Blue bars are indicating the computing time used by a naive C++ implementation.
Green bars are indicating the computing time used by M. Tanaka's Ruby-extension *NArray*.
Red bars are indicating the computing time used by the JIT-based implementation of *HornetsEye*.

Environment

- 1.2 GHz AMD Duron™ processor
- g++ 4.1.3 compiler
- ruby 1.8.6 interpreter
- libjit-0.1.1 just-in-time compiler

Optimisation Potential

- cache for reusing compiled methods
- loop unrolling