Yayi

Raffi Enficiaud

1 History & background

We are all lazy people: we want the algorithms we develop run on *everything* we need, although we do not know what we need beforehand. This is why *genericity* is so appealing: develop once, run *possibly* on many "things". Algorithmic genericity states in first place the dissociation between the internal representation of data structures and the algorithms that run on them, which seems quite straightforward. However, after having developed our first generic algorithm, we see that this is not so simple: there are several levels of genericity, as well as several means to achieve each of them, and a bit of architectural design is required. Object Oriented Programming and template meta-programming are both powerful tools for isolating the concepts involved in Image Processing and Mathematical Morphology, and seem to be a good start. More than that, it turns out that meta-programming is also an efficient way of providing the end user with both genericity and computational efficiency.

Some of the pioneering ideas were presented by Darbon *et al.*[4], in which the relevance of using iterators for image processing (among other concepts,) is emphasized. This intermediate layer lead to the abstraction of the image domain logic. In the meanwhile, Romain Lerallut and Raffi Enficiaud started developing "Morph-M"[1] during their PhD. The project initially aimed at being a generic rewrite of XLim3D [2], but soon became a complete framework for developing new algorithms for Mathematical Morphology (see [9] and [5],) in a multidimensional and multispectral manner. The high degree of abstraction lead to several advances in the field ([8], [10].)

Yayi aims at investigating further in the direction of generic meta-programming paradigms, in order to cope with some of the identified limitations of genericty in general. The development of Yayi started in 2007, as an open-source initiative with the non-restrictive Boost licence, mainly conducted by one developer.

2 Design

The main motivation is to provide highly generic and efficient tools for developing algorithms. The second motivation is to provide tools for developing new algorithms quickly and easily.

Yayi uses mainly three functional layers:

- C++ template layer: this is the core of Yayi, which extensively uses the C++ meta-template mechanisms (template forwarding, specializing, compile time computations, etc.)
- C++ compiled layer (library).
- Python bindings: this interface is particularly convenient and suitable as an intermediate algorithmic level, where pixel/neighborhood operations (eg. color processing, dilation/erosions, ...) are used as algorithmic entities

General consideration and architecture

Cross platform It should be possible to compile Yayi on a wide range of platforms without any modification to the original library[2]. Yayi uses plain C++.

1. formerly known as "Morphee"
2. or at least a few to cover some specific issues

Tools & Build system There should always be a free tool for building Yayi, on any platform (and Windows platform should not be an non-addressed exception.) Yayi uses CMake for its build system, and Visual Express/Windows SDK, XCode, gcc, clang... are able to compile it.

Minimal dependencies Yayi depends on a minimal set of tools, either for I/O (PNG, Jpeg, HDF5 (optional), numpy (Python, optional)) or for using relevant C++ structures (Boost C++ library [1]). Python is optional. There is no GUI considerations.

Licence The licence should not be an issue for the *users*. Yayi is licensed under the permissive Boost license. The problem is that this licence is not widely known.

3 Content

Structures

Support classes (most with python bindings)
- variants, generic pixels, generic positions, generic "bounding boxes"
- color information
- graph (directed/undirected), trees
- histograms
- priority queues

Image classes
- Image classes (generic, interface, python)
- Image processor classes
- Image neighborhood processor classes

Structuring elements
- Runtime structuring elements, pair of SE (hit-or-miss), mutable SE
- Compilation-time structuring elements

Functions & Algorithms

Basic image transforms
- Comparisons: $\vee$, $\wedge$, image vs. scalars, image vs. images, with several type of predicates and outputs
- Color transformations: channel composition/extraction, truly bijective to/from XYZ, HLS, Lab, YUV. Proper colour handling through appropriate structures
- Arithmetic ("+", "-", "clamp", "abs", ...,), logic ("&", "||", ...,) mathematical (matrix transform, log, exp, ...)

IO
- PNG, JPeg
- HDF5 (optional)
- Numpy (python/optional)

Basic mathematical morphology
- Erosion, dilations, openings, closings
- Minkowski addition, subtraction
- Geodesic erosions and dilations
- Hit or miss: Soille grey level with flat SE definition (see [11])
- Morphological gradients: thick, upper/lower half
- Algebraic opening/closing (in progress)

Reconstruction algorithms

- Morphological reconstructions (opening and closing), any dimensional on any ordered type
- Fill holes
- h-minima/maxima
- Levellings (definition of Gomila [7])

Measurements over image or non overlapping regions

- Mean, variance, median
- Min/max
- Histograms
- Circular means and variances

Local processing (with a structuring element)

- Local mean, median
- Local circular (weighted/non weighted) mean and variance

Distance algorithms

- Exact distance transform in any dimension [5]
- Geodesic, grid distances
- Color distance functions (definitions in YUV, Lab/Lab hue, HLS, hue... spaces)
- Morphological quasi-distance [6], [3]

Labelling

- Binary with/without background
- With measurements computed over the region (min/max, area)
- With/without output of the adjacency graph

Segmentation

- Isotropic implementation of the watershed with priority queues (in any dimension on any ordered type)
- Viscous watershed [5] (not fully functional yet.)

4 Samples

Directional opening of leave images

The function filter_directional_open performs a directional filtering for extracting the venation. For each directions α, it keeps structures that are not thicker than ℓ_1 = max_thickness ($\text{im}_{th_1} = \text{im}_{in} - \gamma_{S_{\ell_1}^{\alpha\perp}}(\text{im}_{in})$) and longer than ℓ_2 = min_length ($\text{im}_{filt} = \gamma_{S_{\ell_2}^{\alpha}}(\text{im}_{th_1})$.)

```python
import math as m
def createLineOriented(l, angle):
  """Creates an oriented segment of length l"""
  return [(trunc(i*m.cos(angle)), trunc(i*m.sin(angle))) for i in range(-l, l
      +1)]

def createLineOrientedSE(l, angle):
  """Creates a SE with length l and orientation angle"""
  listpoints = createLineOriented(l, angle)
  return YAYI.SE.SEFactory(YAYI.SE.e_set_neighborlist, 2, listpoints, YAYI.SE.
      e_sest_neighborlist_generic_single)

def filter_directional_open(iml, number_angles = 12, min_length = 7,
    max_thickness = 5):
  """Directional opening"""
  out = []
  angle = lambda x: m.pi * x / number_angles
  se1 = createLineOrientedSE(max_thickness, current_angle + m.pi/2)
  se2 = createLineOrientedSE(min_length, current_angle)
  for i in range(number_angles):
    current_angle = angle(i)
    th2 = AAbsSub(iml, MOpen(iml, se = se1))
    imot22 = MOpen(th2, se = se2)
    out.append(imot22)
  return out, reduce(lambda x,y: MSup(x,y), out) # list and union
```

Polynomial fitting of the connected components

Performs a polynomial fitting of each connected component (previous image.)

```
def connected_components_polynomial_regression_order2(im, se = sq2D):
  """Returns a 2-polynomial for each connected component (non black)"""
  list_of_offsets = YAYI.LAB.ImageLabelNonBlackToOffset(im, se)
  out = []
  for index, i in enumerate(list_of_offsets):
    coords  = YAYI.CORE.FromOffsetsToCoordinates(im.Size, i)
    poly1 = fit_2polynomial_in_x(coords) # fitting using numpy
    out.append((poly1, index))
  return out, list_of_offsets
```

References

[1] The Boost C++ librairy. http://www.boost.org.

[2] XLim3D. http://cmm.ensmp.fr/xlim3d.html.

[3] BEUCHER, S. Numerical residues. In *Proceedings of the 7th International Symposium on Mathematical Morphology* (avril 2005), pp. 23–32.

[4] DARBON, J., GÉRAUD, T., AND DURET-LUTZ, A. Generic implementation of morphological image operators. In *International Symposium on Mathematical Morphology VI* (2002), pp. 175–184.

[5] ENFICIAUD, R. *Multi-dimensional and multi-spectral algorithms in the field of Mathematical Morphology : The meta-programming approach.* PhD thesis, Centre de Morphologie Mathématique, École des Mines de Paris, 2007.

[6] ENFICIAUD, R. Queue and priority queue based algorithms for computing the quasi-distance transform. In *Image Analysis and Recognition*, A. Campilho and M. Kamel, Eds., vol. 6111 of *Lecture Notes in Computer Science*. Springer, 2010, pp. 35–44.

[7] GOMILA, C. *Mise en correspondance de partition en vue de suivi d'objets.* PhD thesis, Centre de Morphologie Mathématique, École des Mines de Paris, 2001.

[8] LAVEAU, N., AND BERNARD, C. Structuring elements following the optical flow. In *Proceedings of the 7th International Symposium on Mathematical Morphology* (avril 2005), pp. 43–52.

[9] LERALLUT, R. *Modélisation et Interprétation d'Images à l'aide de Graphes.* PhD thesis, Centre de Morphologie Mathématique, École des Mines de Paris, 2006.

[10] LERALLUT, R., DECENCIÈRE, E., AND MEYER, F. Image filtering using morphological amoebas. In *Proceedings of the 7th International Symposium on Mathematical Morphology* (avril 2005), pp. 13–22.

[11] NAEGEL, B., PASSAT, N., AND RONSE, C. Grey-level hit-or-miss transforms–part i: Unified theory. *Pattern Recognition 40*, 2 (2007), 635–647.