Olena & Milena in a Few Words

EPITA Research and Development Laboratory (LRDE)

May 2009

Outline

1. About Milena
   - Presentation
   - Genericity
   - Comparison
2. Current Status
   - Library
   - Dynamic Interface

Outline

Outline

Project 1/2

Naming

Olena : image processing[a] platform (also project name)

Milena : image processing library = part of Olena

[a] IP, image processing for short

Goals

1. Focus on the library part (Milena)
2. Add a scripting layer (interpreted environment).
3. Add extra tools
   (visual env., interface with The GIMP, Octave, etc.)

Project 2/2

Rational

Features: platform features come from the library

Limitations: library limitations are viral:
they affect the platform

A Couple of Key Ideas

Operators: too many things in IP (algorithms, methods...)

Objectives: instead, to ease programming IP

What's In a Library

Algorithms:
procedures dedicated to image processing and pattern recognition

Data types for pixel values:
gray level types with different quantizations, several floating types, color types

Data structures:
for instance, many ways to define images and sets of points

A lot of auxiliary tools:
they help to easily write readable algorithms and methods in a concise way!

Objectives of Milena as a Feature List

Genericity	not limited to very few types of values and images
Simplicity	as easy to use as a C or Java library
Efficiency	ready to intensive computation (large data / sets of data)
Composability	coherency of tools ensure software building from blocks
Safety	errors are pointed out at compile-time, otherwise at run-time
Reusability	software blocks are provided for general purpose

Getting at the same time all those features is very challenging.

History

	Version	Features	Misfeatures
2000-01	0.1	genericity w.r.t. values	rectangular 2D images only!
2001-04	0.10	genericity w.r.t. both structures and values	limitations... (Cf. next slides)
2004-07	X	prototype	too sophisticated design, very slow compilation :-(yet many solutions used in v1.0 :-)
2007	0.11	just an update of 0.10	same as 0.10
2007-09	1.0	full genericity	...

Outline

The Most Dummy Example

Filling an image ima with the value v:

```
// Java or C -like code

void fill(image* ima, unsigned char v)
{
  for (int i = 0; i < ima->nrows; ++i)
    for (int j = 0; j < ima->ncols; ++j)
      ima->data[i][j] = v;
}
```

Note that we really have here an example very representative of an algorithm and of many pieces of existing code.

Some Observations 1/2

Kleenex

There are a lot of implicit assumptions about the input:

- The input image has to be 2D;
- its definition domain has to be a rectangle;
- this rectangle shall start at (0,0);
- data cannot be of a different type than "unsigned char";
- last, data need to be stored as a 2D array in RAM.

This is a kleenex code:

"code once, run on one image type"

For instance this routine cannot work on a region of interest of a 2D image having floating values.

Some Observations 2/2

Obfuscation

Working on a particular type of image leads to the presence of implementation details.

This is a dirty kleenex code:

“implementation details obfuscate the actual algorithm”

Furthermore, it is:

- verbose
- error-prone
- hard to maintain.

Definition

A Generic Algorithm

A generic algorithm is written once (without duplicates)

and

works on different kind of input

Generic algorithm translation

Algorithm:

```
Procedure fill
   ima : an image (type: any type I)
   v   : a value  (type: value type of I)
begin
   for all p in ima domain
      ima(p) ← v
end
```

// Milena code:

```
template <typename I>
void fill( I& ima,
           mln_value(I) v )
{
  mln_piter(I) p(ima.domain());
  for_all(p)
    ima(p) = v;
}
```

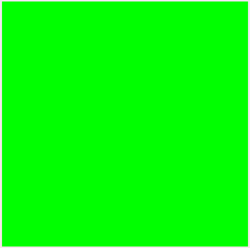
Example

The basic (common) run:

```
using literal::green;
data::fill(lena, literal::green);
```

before:

after:



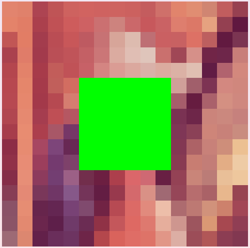
Example cont'd

Filling only a region of interest (a set of points):

```
mln_VAR(roi, lena | make::box2d(5,5, 10,10));
data::fill(roi, literal::green);
```

before:

after:



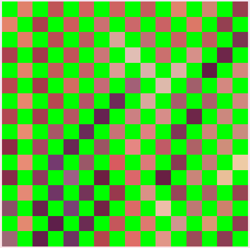
Example cont'd

Filling only points verifying a predicate:

```
mln_VAR(lena_c, lena | fun::p2b::chess());
data::fill(lena_c, literal::green);
```

before:

after:



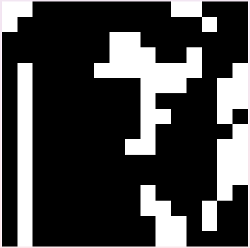
Example cont'd

Likewise, the predicate being a mask image:

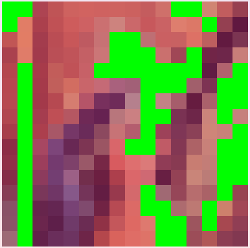
```
mln_VAR(lena_m, lena | pw::value(mask));
data::fill(lena_m, literal::green);
```

before:

mask:



after:



Example cont'd

Likewise, relying on an image of labels:

```
mln_VAR(lena_3, lena | (pw::value(label) == 3));
data::fill(lena_3, literal::green);
```

before:

label:

after:

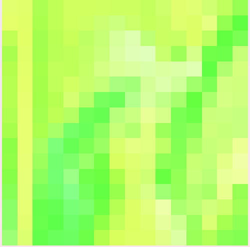
Example cont'd

Filling only a component:

```
mln_VAR(lena_g, fun::access::green << lena);
data::fill(lena_g, literal::green);
```

before:

after:



Example cont'd

Mixing several "image views":

```
mln_VAR(lena_g3, lena_g | pw::value(label) == 3);
data::fill(lena_g3, literal::green);
```

before:

label:

after:

Some Remarks 1/2

Replace the 2D image by:

- a signal
- a volume
- a graph
- a complex
- etc.

and it works as is...

Some Remarks 2/2

Genericity applies on:

- values of images
- structures of images
- modifiers of images (Cf. previous slides)
- neighborhoods
- functions
- etc.

Past Limitations

From 0.11 to 1.0

Limitations of version 0.11 did not allow to have the previous examples work.

Outline

Four Kinds of Users

- **Assemblers**: just compose components (algorithms) to solve a problem
- **Designers**: write new algorithms
- **Providers**: write new data types
- **Architects**: focus on the library core

Required skills go increasingly within this list.

Code Comparison

Image practionners write algorithms...

...so have a look at the same code.

Tivoli

emacs@test

File Edit Options Buffers Tools C Help

```c
// http://www.tsi.enst.fr/~fouquier/tivoli/

Volume* numDilate(Volume* volume, int connectivity)
{
  Volume* dilated;
  long int *oB, oFP, oPbL, oLbS;
  int ix, nx, iy, ny, iz, nz, i;
  U8BIT_t *ptVolume, *ptDilated, *pt, max;

  setBorderWidth ( volume, -1 );
  oB = offsetBox ( volume, connectivity );
  oFP = offsetFirstPoint ( volume );
  ptVolume = data_U8BIT ( volume ) + oFP;
  dilated = duplicateVolumeStructure ( volume, "dilated" );
  ptDilated = data_U8BIT ( dilated ) + oFP;

  setBorderLevel ( volume, 0 );
  getSize ( volume, &nx, &ny, &nz );
  oPbL = offsetPointBetweenLine ( volume );
  oLbS = offsetLineBetweenSlice ( volume );
  for ( iz = 0; iz < nz; iz++ )
    {
      for ( iy = 0; iy < ny; iy++ )
        {
          for ( ix = 0; ix < nx; ix++ )
            {
              pt = ptVolume++;
              max = *pt;
              for ( i = 0; i < connectivity; i++ )
                {
                  pt += oB[i];
                  if ( *pt > max )
                    max = *pt;
                }
              *ptDilated++ = max;
            }
          ptVolume += oPbL;
          ptDilated += oPbL;
        }
      ptVolume += oLbS;
      ptDilated += oLbS;
    }

  free ( oB );
  return ( dilated );
}
```

--:** tivoli.c Top L47 (C/l Abbrev)

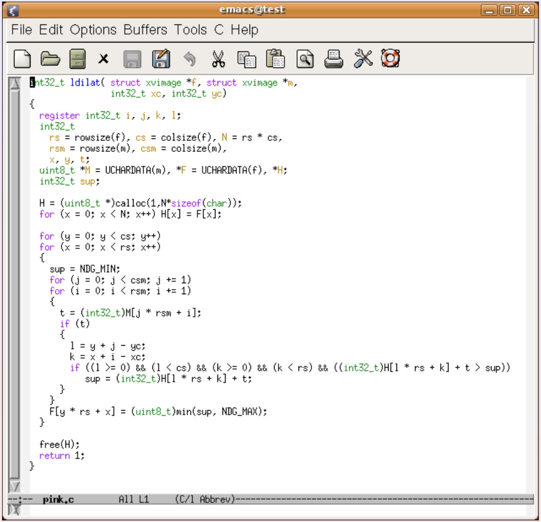
Context: TSI, ENST

Author: theo

Year: 1995

Language: C

Pink



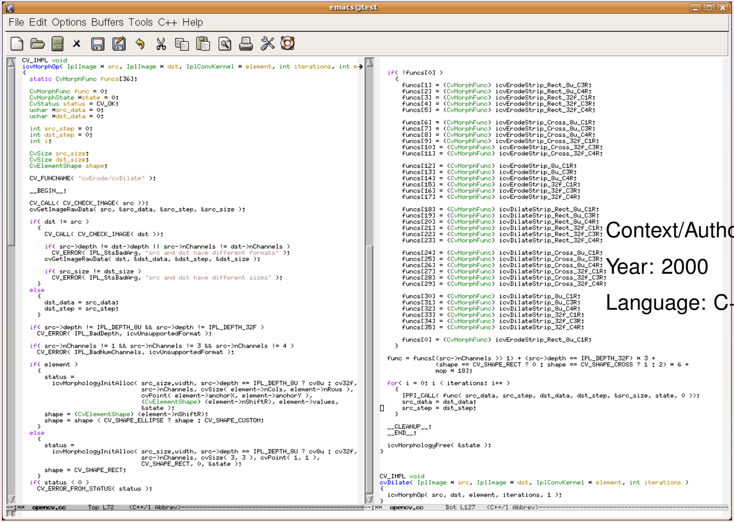
Context: ESIEE

Author: Michel Couprie

Year: 1997

Language: C

OpenCV



Context/Author: Intel

Year: 2000

Language: C++

ITK 1/2

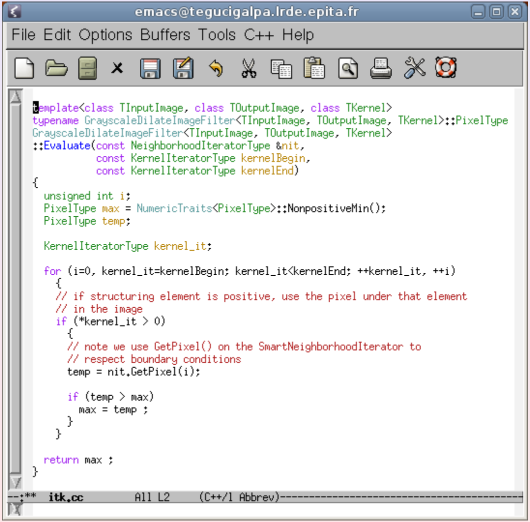
```cpp
template< class ImageType >
void dilationGrayscale(
  const std::string & inputFileName,
  const std::string & outputFileName,
  const std::vector<unsigned int> & radius,
  const std::string & boundaryCondition )
{
  /** Typedefs. */
  typedef typename ImageType::PixelType                      PixelType;
  const unsigned int Dimension = ImageType::ImageDimension;
  typedef itk::ImageFileReader< ImageType >                  ReaderType;
  typedef itk::ImageFileWriter< ImageType >                  WriterType;
  typedef itk::BinaryBallStructuringElement<
    PixelType, Dimension >                                   StructuringElementType;
  typedef typename StructuringElementType::RadiusType RadiusType;
  typedef itk::GrayscaleDilateImageFilter<
    ImageType, ImageType, StructuringElementType >           DilateFilterType;
  typedef typename
    DilateFilterType::DefaultBoundaryConditionType           BoundaryConditionType;

  /** Declarations. */
  typename ReaderType::Pointer reader = ReaderType::New();
  typename WriterType::Pointer writer = WriterType::New();
  typename DilateFilterType::Pointer dilation = DilateFilterType::New();

  /** Setup the reader. */
  reader->SetFileName( inputFileName.c_str() );

  /** Set a boundary condition value.
   * This is the value outside the image.
   * By default it is set to min(PixelType).
   */
  BoundaryConditionType bc;
  PixelType bcValue = itk::NumericTraits<PixelType>::NonpositiveMin();
  if ( boundaryCondition != "")
  {
    if ( itk::NumericTraits<PixelType>::is_integer )
    {
      bcValue = static_cast<PixelType>( atoi( boundaryCondition.c_str() ) );
    }
    else
    {
      bcValue = static_cast<PixelType>( atof( boundaryCondition.c_str() ) );
    }
    bc.SetConstant( bcValue );
    dilation->OverrideBoundaryCondition(&bc);
  }

  /** Create the structuring element. */
  RadiusType  radiusarray;
  for ( unsigned int i = 0; i < Dimension; i++ )
  {
    radiusarray.SetElement( i, radius[ i ] );
  }
  StructuringElementType  S_ball;
  S_ball.SetRadius( radiusarray );
  S_ball.CreateStructuringElement();

  /** Setup the dilation filter. */
  dilation->SetKernel( S_ball );
  dilation->SetInput( reader->GetOutput() );

  /** Write the output image. */
  writer->SetFileName( outputFileName.c_str() );
--:** itk.cc        Top L1     (C++/l Abbrev)----
```

```cpp
  writer->SetInput( dilation->GetOutput() );
  writer->Update();

} // end dilationGrayscale()



template<class TInputImage, class TOutputImage, class TKernel>
class ITK_EXPORT GrayscaleDilateImageFilter :
    public MorphologyImageFilter<TInputImage, TOutputImage, TKernel>
{
public:
  /** Standard class typedefs. */
  typedef GrayscaleDilateImageFilter Self;
  typedef MorphologyImageFilter<TInputImage, TOutputImage, TKernel>
  Superclass;
  typedef SmartPointer<Self>        Pointer;
  typedef SmartPointer<const Self>  ConstPointer;

  /** Standard New method. */
  itkNewMacro(Self);

  /** Runtime information support. */
  itkTypeMacro(GrayscaleDilateImageFilter,
               MorphologyImageFilter);

  /** Declaration of pixel type. */
  typedef typename Superclass::PixelType PixelType;

  /** Kernel (structuring element) iterator. */
  typedef typename Superclass::KernelIteratorType  KernelIteratorType;

  /** Neighborhood iterator type. */
  typedef typename Superclass::NeighborhoodIteratorType NeighborhoodIteratorType ;

  /** Kernel typedef. */
  typedef typename Superclass::KernelType KernelType;

  /** Default boundary condition type */
  typedef typename Superclass::DefaultBoundaryConditionType DefaultBoundaryConditionType;

  /** ImageDimension constants */
  itkStaticConstMacro(InputImageDimension, unsigned int,
                      TInputImage::ImageDimension);
  itkStaticConstMacro(OutputImageDimension, unsigned int,
                      TOutputImage::ImageDimension);
  itkStaticConstMacro(KernelDimension, unsigned int,
                      TKernel::NeighborhoodDimension);

#ifdef ITK_USE_CONCEPT_CHECKING
  /** Begin concept checking */
  itkConceptMacro(InputConvertibleToOutputCheck,
    (Concept::Convertible<PixelType, typename TOutputImage::PixelType>));
  itkConceptMacro(SameDimensionCheck1,
     (Concept::SameDimension<InputImageDimension, OutputImageDimension>));
  itkConceptMacro(SameDimensionCheck2,
    (Concept::SameDimension<InputImageDimension, KernelDimension>));
  itkConceptMacro(InputGreaterThanComparableCheck,
    (Concept::GreaterThanComparable<PixelType>));
  itkConceptMacro(KernelGreaterThanIntCheck,
    (Concept::GreaterThanComparable<typename TKernel::PixelType, int>));
  /** End concept checking */
#endif
--:** itk.cc        34% L69    (C++/l Abbrev)----
```

```cpp
protected:
  GrayscaleDilateImageFilter();
  ~GrayscaleDilateImageFilter() {};

  /** Evaluate image neighborhood with kernel to find the new value
   * for the center pixel value
   *
   * It will return the maximum value of the image pixels whose corresponding
   * element in the structuring element is positive. This version of
   * Evaluate is used for non-boundary pixels. */
  PixelType Evaluate(const NeighborhoodIteratorType &nit,
                     const KernelIteratorType kernelBegin,
                     const KernelIteratorType kernelEnd);

private:
  GrayscaleDilateImageFilter(const Self&); //purposely not implemented
  void operator=(const Self&); //purposely not implemented

  // Default boundary condition for dilation filter, defaults to
  // NumericTraits<PixelType>::NonpositiveMin()
  DefaultBoundaryConditionType m_DilateBoundaryCondition;

} ; // end of class

template<class TInputImage, class TOutputImage, class TKernel>
GrayscaleDilateImageFilter<TInputImage, TOutputImage, TKernel>
::GrayscaleDilateImageFilter()
{
  m_DilateBoundaryCondition.SetConstant( NumericTraits<PixelType>::NonpositiveMin() );
  this->OverrideBoundaryCondition( &m_DilateBoundaryCondition );
}

template<class TInputImage, class TOutputImage, class TKernel>
typename GrayscaleDilateImageFilter<TInputImage, TOutputImage, TKernel>::PixelType
GrayscaleDilateImageFilter<TInputImage, TOutputImage, TKernel>
::Evaluate(const NeighborhoodIteratorType &nit,
           const KernelIteratorType kernelBegin,
           const KernelIteratorType kernelEnd)
{
  unsigned int i;
  PixelType max = NumericTraits<PixelType>::NonpositiveMin();
  PixelType temp;

  KernelIteratorType kernel_it;

  for (i=0, kernel_it=kernelBegin; kernel_it<kernelEnd; ++kernel_it, ++i)
    {
    // if structuring element is positive, use the pixel under that element
    // in the image
    if (*kernel_it > 0)
      {
      // note we use GetPixel() on the SmartNeighborhoodIterator to
      // respect boundary conditions
      temp = nit.GetPixel(i);

      if (temp > max)
        max = temp ;
      }
    }

  return max ;
}
--:** itk.cc        Bot L169   (C++/l Abbrev)----
```

Find file: ~/lrde/olena/presentations/olena.2009.esiee/code/

ITK 2/2




```
template<class TInputImage, class TOutputImage, class TKernel>
typename GrayscaleDilateImageFilter<TInputImage, TOutputImage, TKernel>::PixelType
GrayscaleDilateImageFilter<TInputImage, TOutputImage, TKernel>
::Evaluate(const NeighborhoodIteratorType &nit,
           const KernelIteratorType kernelBegin,
           const KernelIteratorType kernelEnd)
{
  unsigned int i;
  PixelType max = NumericTraits<PixelType>::NonpositiveMin();
  PixelType temp;

  KernelIteratorType kernel_it;

  for (i=0, kernel_it=kernelBegin; kernel_it<kernelEnd; ++kernel_it, ++i)
    {
    // if structuring element is positive, use the pixel under that element
    // in the image
    if (*kernel_it > 0)
      {
      // note we use GetPixel() on the SmartNeighborhoodIterator to
      // respect boundary conditions
      temp = nit.GetPixel(i);

      if (temp > max)
        max = temp ;
      }
    }

  return max ;
}
```

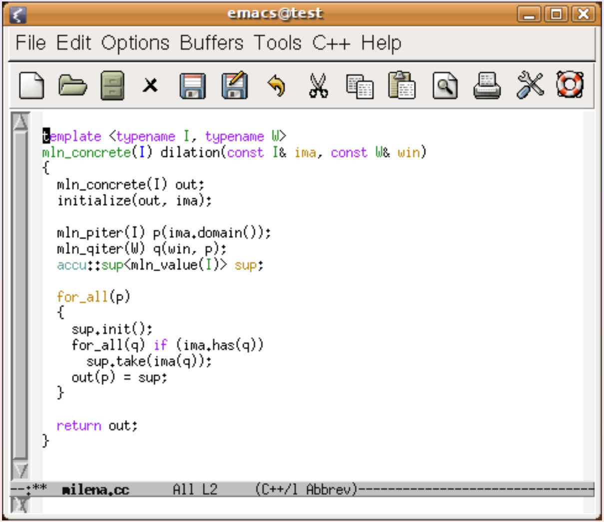
Context: ITK

Author: Insight Software Consortium

Year: 2006

Language: C++

Milena



Context: LRDE

Author: theo

Year: 2007

Language: C++

Outline

Some Facts

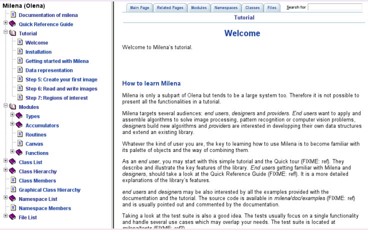
About versions:

- 1.0β released in December 2008
- 1.0 is due to June 10th, 2009

Current version is fully functional and used:

- in large projects:
  - Melimage (funded by INCA)
  - SCRIBO (funded by System@tic)
- in students projects
  - about a dozen per years

Documentation


Milena (Olena)
Documentation of milena
Quick Reference Guide
Tutorial
Welcome
Installation
Getting started with Milena
Data representation
Step 5: Create your first image
Step 6: Read and write images
Step 7: Regions of interest
Modules
Types
Accumulators
Routines
Canvas
Functions
Class List
Class Hierarchy
Class Members
Graphical Class Hierarchy
Namespace List
Namespace Members
File List

Main Page | Related Pages | Modules | Namespaces | Classes | Files | Search for

Tutorial

Welcome

Welcome to Milena's tutorial.

How to learn Milena

Milena is only a subpart of Olena but tends to be a large system too. Therefore it is not possible to present all the functionalities in a tutorial.

Milena targets several audiences: *end users*, *designers* and *providers*. *End users* want to apply and assemble algorithms to solve image processing, pattern recognition or computer vision problems, *designers* build new algorithms and *providers* are interested in developping their own data structures and extend an existing library.

Whatever the kind of user you are, the key to learning how to use Milena is to become familiar with its palette of objects and the way of combining them.

As an *end user*, you may start with this simple tutorial and the Quick tour (FIXME: ref). They describe and illustrate the key features of the library. *End users* getting familiar with Milena and *designers*, should take a look at the Quick Reference Guide (FIXME: ref). It is a more detailed explanations of the library's features.

end users and *designers* may be also interested by all the examples provided with the documentation and the tutorial. The source code is available in *milena/doc/examples* (FIXME: ref) and is usually pointed out and commented by the documentation.

Taking a look at the test suite is also a good idea. The tests usually focus on a single functionality and handle several use cases which may overlap your needs. The test suite is located at

We have

- a white paper
- a tutorial
- a reference guide

`http://www.lrde.epita.fr/dload/doc/milena-1.0/`

Entering Milena

Easy? Quick?

From our experiments:

- two days are enough to take Milena in hand
- the learning curve is great.

Outline

Static-Dynamic Bridge

Need for a Bridge

On one hand:
Milena = efficient C++ generic, thus **static**, code.

On the other hand:
a **dynamic** environment (script, interpreter, GUI).

$\Rightarrow$ A bridge between both worlds is required.

Our Solution: Swilena 1/2

Tools

Swilena is the bridge provided in Olena to access Milena from another language.

SPS (Swilena Python Shell) is a command line interpreter.

History:

- architecture sketched in 2000 (GCSE Workshop)
- started in 2002
- functional until version 0.11
- up again in Summer 2008

Our Solution: Swilena 2/2

The how-to

- it works on closed world (a context)
- for a given type, you get access to a subset of the library
  (for instance, `image2d<int_u8>`)

About writing this bridge

- the starting cost is very quickly amortized
- it can be done in a very modularized way

Sample Code 1/3

Morphological glue:

```
%module morpho

%include "concrete.ixx"

/* dilation */
%{
#include "mln/morpho/dilation.hh"
%}
%include "mln/morpho/dilation.hh"
%define instantiate_dilation(Name, I, W)
  %template() mln::trait::concrete< I >;
  %template(Name) mln::morpho::dilation< I, W >;
%enddef

/* morphology */
%define instantiate_morpho(I, W, N)
  instantiate_dilation(dilation, I, W)
  instantiate_erosion(erosion, I, W)
  /* ... */
%enddef
```

Sample Code 2/3

A precise world:

```
%module image2d_int

%include "intp.ixx"

%include "image2d.ixx"
instantiate_image2d(image2d_int, int)

%include "window2d.ixx"
%include "neighb2d.ixx"

%include "morpho.ixx"
instantiate_morpho(mln::image2d<int>, mln::window2d, mln::neighb2d)
```

Sample Code 3/3

Sample use:

```
from swilena import *

# Module alias.
image = image2d_int_u8

# Load.
f = image.io_pgm_load("lena.pgm")

# Gradient.
g = image.morpho_elementary_gradient(f, c4())

# Area closing of the gradient.
h = image.morpho_closing_area(g, c4(), 50)

# Watershed transform.
n_basins = int_u8();
w = image.morpho_watershed_flooding(h, c4(), nbasins)
print n_basins

# Save.
image.io_pgm_save(w, "w.pgm")
```