Olena & Milena in a Few Words

EPITA Research and Development Laboratory (LRDE)

May 2009

Outline

Outline

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.)

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 <u>very</u> challenging.

	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.

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.

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.

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;
}
```

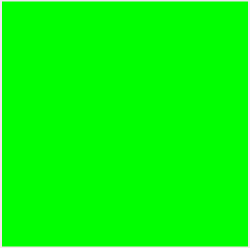
Example

The basic (common) run:

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

before:

after:

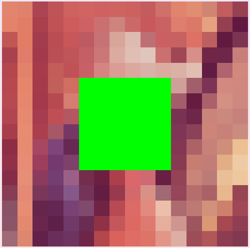


Filling only a region of interest (a set of points):

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

before:

after:

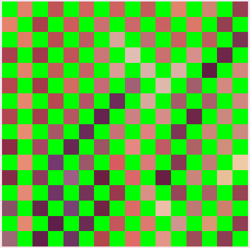


Filling only points verifying a predicate:

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

before:

after:

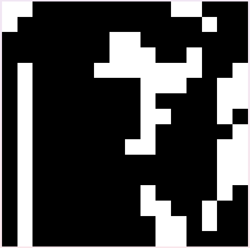


Likewise, the predicate being a mask image:

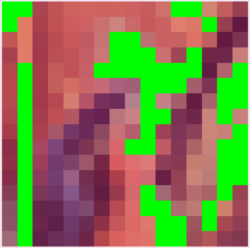
```
mln_VAR(lena_m, lena | pw::value(mask));
data::fill(lena_m, literal::green);
```

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:

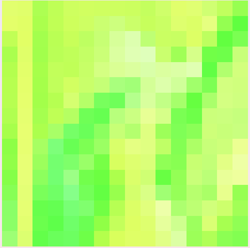
after:

Filling only a component:

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

before:

after:



Mixing several "image views":

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

before:

label:

after:

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)---

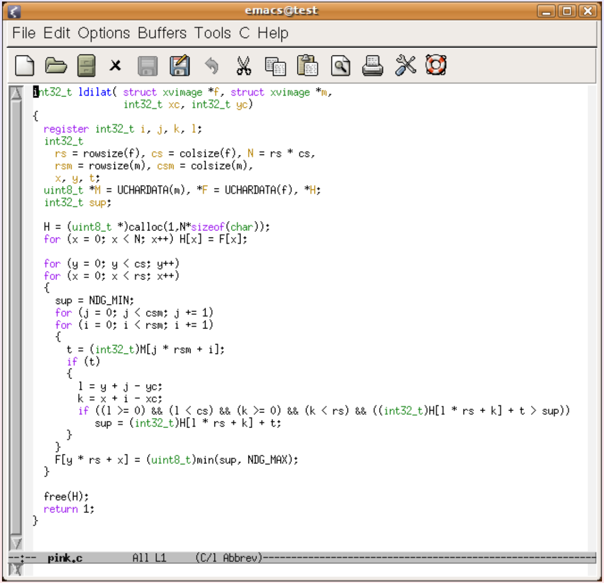
Context: TSI, ENST

Author: theo

Year: 1995

Language: C

Pink



Context: ESIEE

Author: Michel Couprie

Year: 1997

Language: C

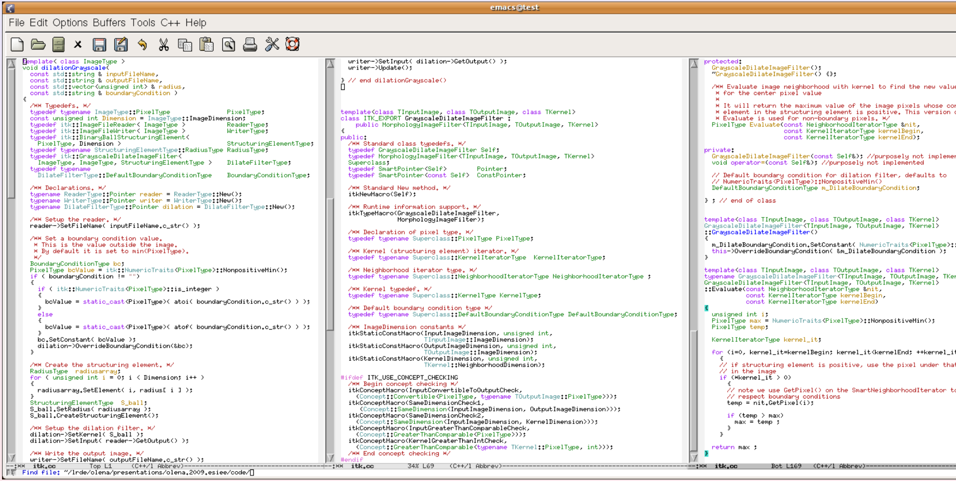
OpenCV

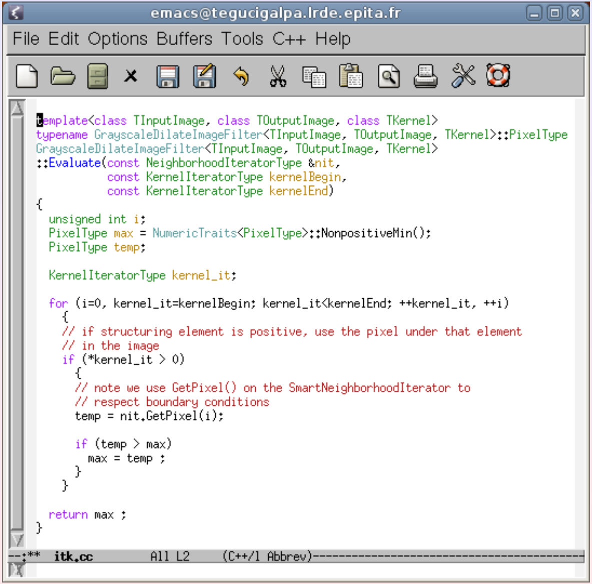
```
emacs@test
File Edit Options Buffers Tools C++ Help

CV_IMPL void
icvMorphOp( IplImage * src, IplImage * dst, IplConvKernel * element, int iterations, int m
{
  static CvMorphFunc funcs[36];

  CvMorphFunc func = 0;
  CvMorphState *state = 0;
  CvStatus status = CV_OK;
  uchar *src_data = 0;
  uchar *dst_data = 0;

  int src_step = 0;
  int dst_step = 0;
  int i;

  CvSize src_size;
  CvSize dst_size;
  CvElementShape shape;

  CV_FUNCNAME( "cvErode/cvDilate" );

  __BEGIN__;

  CV_CALL( CV_CHECK_IMAGE( src ));
  cvGetImageRawData( src, &src_data, &src_step, &src_size );

  if( dst != src )
    {
      CV_CALL( CV_CHECK_IMAGE( dst ));

      if( src->depth != dst->depth || src->nChannels != dst->nChannels )
        CV_ERROR( IPL_StsBadArg, "src and dst have different formats" );
      cvGetImageRawData( dst, &dst_data, &dst_step, &dst_size );

      if( src_size != dst_size )
        CV_ERROR( IPL_StsBadArg, "src and dst have different sizes" );
    }
  else
    {
      dst_data = src_data;
      dst_step = src_step;
    }

  if( src->depth != IPL_DEPTH_8U && src->depth != IPL_DEPTH_32F )
    CV_ERROR( IPL_BadDepth, icvUnsupportedFormat );

  if( src->nChannels != 1 && src->nChannels != 3 && src->nChannels != 4 )
    CV_ERROR( IPL_BadNumChannels, icvUnsupportedFormat );

  if( element )
    {
      status =
        icvMorphologyInitAlloc( src_size.width, src->depth == IPL_DEPTH_8U ? cv8u : cv32f,
                                src->nChannels, cvSize( element->nCols, element->nRows ),
                                cvPoint( element->anchorX, element->anchorY ),
                                (CvElementShape) (element->nShiftR), element->values,
                                &state );
      shape = (CvElementShape) (element->nShiftR);
      shape = shape < CV_SHAPE_ELLIPSE ? shape : CV_SHAPE_CUSTOM;
    }
  else
    {
      status =
        icvMorphologyInitAlloc( src_size.width, src->depth == IPL_DEPTH_8U ? cv8u : cv32f,
                                src->nChannels, cvSize( 3, 3 ), cvPoint( 1, 1 ),
                                CV_SHAPE_RECT, 0, &state );
      shape = CV_SHAPE_RECT;
    }
  if( status < 0 )
    CV_ERROR_FROM_STATUS( status );

--:** opencv.cc      Top L72    (C++/l Abbrev)----

  if( !funcs[0] )
    {
      funcs[1] = (CvMorphFunc) icvErodeStrip_Rect_8u_C3R;
      funcs[2] = (CvMorphFunc) icvErodeStrip_Rect_8u_C4R;
      funcs[3] = (CvMorphFunc) icvErodeStrip_Rect_32f_C1R;
      funcs[4] = (CvMorphFunc) icvErodeStrip_Rect_32f_C3R;
      funcs[5] = (CvMorphFunc) icvErodeStrip_Rect_32f_C4R;

      funcs[6] = (CvMorphFunc) icvErodeStrip_Cross_8u_C1R;
      funcs[7] = (CvMorphFunc) icvErodeStrip_Cross_8u_C3R;
      funcs[8] = (CvMorphFunc) icvErodeStrip_Cross_8u_C4R;
      funcs[9] = (CvMorphFunc) icvErodeStrip_Cross_32f_C1R;
      funcs[10] = (CvMorphFunc) icvErodeStrip_Cross_32f_C3R;
      funcs[11] = (CvMorphFunc) icvErodeStrip_Cross_32f_C4R;

      funcs[12] = (CvMorphFunc) icvErodeStrip_8u_C1R;
      funcs[13] = (CvMorphFunc) icvErodeStrip_8u_C3R;
      funcs[14] = (CvMorphFunc) icvErodeStrip_8u_C4R;
      funcs[15] = (CvMorphFunc) icvErodeStrip_32f_C1R;
      funcs[16] = (CvMorphFunc) icvErodeStrip_32f_C3R;
      funcs[17] = (CvMorphFunc) icvErodeStrip_32f_C4R;

      funcs[18] = (CvMorphFunc) icvDilateStrip_Rect_8u_C1R;
      funcs[19] = (CvMorphFunc) icvDilateStrip_Rect_8u_C3R;
      funcs[20] = (CvMorphFunc) icvDilateStrip_Rect_8u_C4R;
      funcs[21] = (CvMorphFunc) icvDilateStrip_Rect_32f_C1R;
      funcs[22] = (CvMorphFunc) icvDilateStrip_Rect_32f_C3R;
      funcs[23] = (CvMorphFunc) icvDilateStrip_Rect_32f_C4R;

      funcs[24] = (CvMorphFunc) icvDilateStrip_Cross_8u_C1R;
      funcs[25] = (CvMorphFunc) icvDilateStrip_Cross_8u_C3R;
      funcs[26] = (CvMorphFunc) icvDilateStrip_Cross_8u_C4R;
      funcs[27] = (CvMorphFunc) icvDilateStrip_Cross_32f_C1R;
      funcs[28] = (CvMorphFunc) icvDilateStrip_Cross_32f_C3R;
      funcs[29] = (CvMorphFunc) icvDilateStrip_Cross_32f_C4R;

      funcs[30] = (CvMorphFunc) icvDilateStrip_8u_C1R;
      funcs[31] = (CvMorphFunc) icvDilateStrip_8u_C3R;
      funcs[32] = (CvMorphFunc) icvDilateStrip_8u_C4R;
      funcs[33] = (CvMorphFunc) icvDilateStrip_32f_C1R;
      funcs[34] = (CvMorphFunc) icvDilateStrip_32f_C3R;
      funcs[35] = (CvMorphFunc) icvDilateStrip_32f_C4R;

      funcs[0] = (CvMorphFunc) icvErodeStrip_Rect_8u_C1R;
    }

  func = funcs[(src->nChannels >> 1) + (src->depth == IPL_DEPTH_32F) * 3 +
               (shape == CV_SHAPE_RECT ? 0 : shape == CV_SHAPE_CROSS ? 1 : 2) * 6 +
               mop * 18];

  for( i = 0; i < iterations; i++ )
    {
      IPPI_CALL( func( src_data, src_step, dst_data, dst_step, &src_size, state, 0 ));
      src_data = dst_data;
      src_step = dst_step;
    }

  __CLEANUP__;
  __END__;

  icvMorphologyFree( &state );
}


CV_IMPL void
cvDilate( IplImage * src, IplImage * dst, IplConvKernel * element, int iterations )
{
  icvMorphOp( src, dst, element, iterations, 1 );
}
--:** opencv.cc      Bot L127   (C++/l Abbrev)----
```

Context/Author: Intel

Year: 2000

Language: C++


emacs@test
File Edit Options Buffers Tools C++ Help
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)
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)
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 cor
* element in the structuring element is positive. This version o
* Evaluate is used for non-boundary pixels. */
PixelType Evaluate(const NeighborhoodIteratorType &nit,
const KernelIteratorType kernelBegin,
const KernelIteratorType kernelEnd);
private:
GrayscaleDilateImageFilter(const Self&); //purposely not implemen
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>::
this->OverrideBoundaryCondition(&m_DilateBoundaryCondition);
}
template<class TInputImage, class TOutputImage, class TKernel>
typename GrayscaleDilateImageFilter<TInputImage, TOutputImage, TKer
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
{
// if structuring element is positive, use the pixel under that
// 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/


emacs@tegucigalpa.lrde.epita.fr

File Edit Options Buffers Tools C++ Help

```
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 All L2 (C++/l Abbrev)--

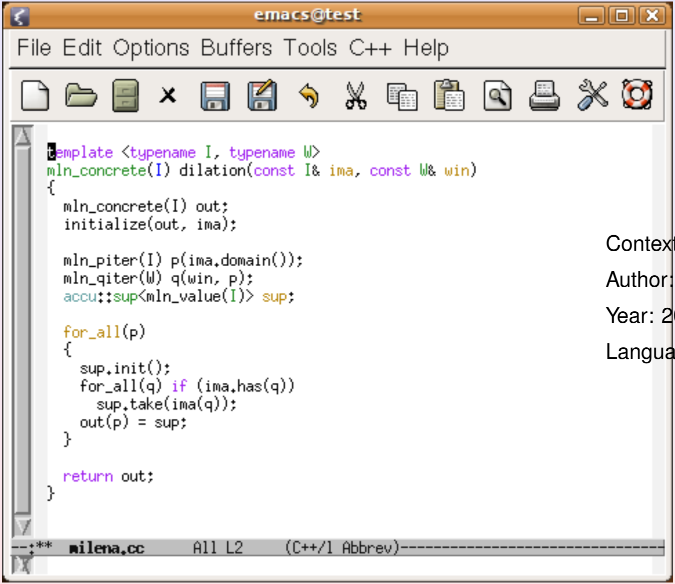
Context: ITK

Author: Insight Software Consortium

Year: 2006

Language: C++

Milena



Context: LRDE

Author: theo

Year: 2007

Language: C++

Outline

Some Facts

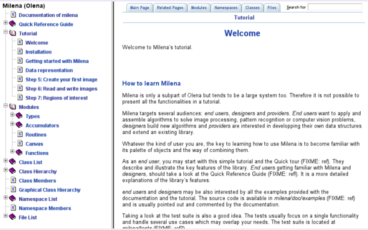
About versions:

- 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



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

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 use:

```python
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")
```