SMIL : Simple Morphological Image Library

Matthieu Faessel, Michel Bilodeau

Centre de Morphologie Mathématique, Mathématiques et Systèmes, MINES ParisTech

Séminaire LRDE, 27/03/2013
# Table of contents

1. Introduction
2. SMIL: General presentation
3. Optimizations
4. Benchmarks/Perspectives
1 Introduction

2 SMIL: General presentation

3 Optimizations

4 Benchmarks/Perspectives
### CMM image libraries history

<table>
<thead>
<tr>
<th>Year</th>
<th>Micromorph (DOS, 16 bits)</th>
<th>Xlim3D (SUN, 32 bits)</th>
<th>Morph-M</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>Micromorph</td>
<td>Xlim3D</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>Mamba</td>
<td>Xlim3D</td>
<td>Morph-M</td>
</tr>
<tr>
<td>2010</td>
<td>Mamba</td>
<td>Xlim3D</td>
<td>Morph-M</td>
</tr>
</tbody>
</table>
Morph-M: a research library

Advantages: suitable for exploration
- Generic (gray, color, multispectral images, graphs, ...)
- N-Dimensions
- Meta-programming
- A lot of contributions (common workspace for students, researchers, ...)

Drawbacks
- Development time
- Clarity of the code (for users and contributors): a high level of abstraction
- **Performances**: not in the initial specifications, very difficult to integrate afterwards
- Proprietary Licence: not very flexible for industrial projects
Morph-M: a research library

Advantages: suitable for exploration
- Generic (gray, color, multispectral images, graphs, ...)
- N-Dimensions
- Meta-programming
- A lot of contributions (common workspace for students, researchers, ...)

Drawbacks
- Development time
- Clarity of the code (for users and contributors): a high level of abstraction
- Performances: not in the initial specifications, very difficult to integrate afterwards
- Proprietary Licence: not very flexible for industrial projects
## New constraints

### Industrial projects with performance constraints increasingly strong

- Real-time applications (ex: industrial control, video surveillance)
- Near real-time processing of large 2D images (ex: 40,000 × 1,200/sec)
- 3D images (ex: 1,000^3 → 1 GPix)
- Embedded systems

### New needs

- Very fast algorithms / Support for large data
- Relatively light
- Easy prototyping
- Ease of integration
New constraints

Industrial projects with performance constraints increasingly strong

- Real-time applications (ex: industrial control, video surveillance)
- Near real-time processing of large 2D images (ex: 40,000 × 1,200/sec)
- 3D images (ex: 1,000³ → 1 GPix)
- Embedded systems

New needs

- Very fast algorithms / Support for large data
- Relatively light
- Easy prototyping
- Ease of integration
Fast Morph-M alternatives

Mamba

Pros:
✓ Fast (intrinsic SIMD functions)
✓ Easy to use (python, embedded viewer, . . . )
✓ Implementation of new python functions is simple

Cons:
✗ Most functions are written in python (with dependency with other python libraries) ⇒ Problems for integration
✗ C based Core: no factorization (much redundant code) ⇒ Rigidity, not suitable for general improvements or newer developments
✗ SIMD intrinsic functions: a lot of code lines
Fast Morph-M alternatives

**Fulguro**

Pros:
- ✔ Fast (intrinsic SIMD functions)
- ✔ C-style factorization (macros)

Cons:
- ✗ C-style factorization (macros)
- ✗ SIMD intrinsic functions: a lot of code lines

**Fast-MorphM**
MorphM SIMD Addon
Very few functions
A solution?

- Genericity
  - Code factorization

- Speed
  - Code redundancy

- Ease of use and development
  - Portability
A new done: Auto-Vectorization

GCC 4.2 new feature: auto-vectorization... reconciles C++ and SIMD

⇒ Start a new library from scratch, using auto-vectorization
Vectorization

SIMD Registers

- 2x doubles
- 16x bytes
- 8x 16-bit shorts
- 4x 32-bit integers
- 2x 64-bit integers
- 1x 128-bit(!) integer

Matthieu Faessel, Michel Bilodeau

SMIL : Simple Morphological Image Library
Vectorization

for(i=0;i<=MAX;i++)
c[i]=a[i]+b[i];

Not vectorized

e.g. 3 x 32-bit unused integers
for(i=0;i<=MAX;i++)
c[i]=a[i]+b[i];

Vectorized
How to vectorize?

- Until auto-vectorization: Intrinsic implementation
  - Requires aligned and contiguous data
  - 1 implementation/data type
  - 1 implementation/SIMD instruction type (SSE, SSE2, ...)

- With auto-vectorization
  - Requires auto-vectorization capable compiler: GCC (≥4.2), ICC, XLC, CLang, MSVC (≥2012), ...
  - Requires aligned and contiguous data
  - Requires some code conditions: write vectorizer-friendly code:
    - Countable loop
    - Avoid aliasing problems (single entry and single exit)
    - Straight-line code
    - The innermost loop of a nest
    - No function calls
Auto-vectorization Example

```c
#define N 128
1. int a[N], b[N];
2. void foo (void)
3. {
4.   int i;
5.   for (i = 0; i < N; i++)
6.     a[i] = i;
7.   for (i = 0; i < N; i+=5)
8.     b[i] = i;
9. }
```

Matthieu Faessel, Michel Bilodeau

SMIL: Simple Morphological Image Library
void supLine(const PIXEL *pIn1, const PIXEL *pIn2, PIXEL *pOut, int count){
    for (int i=0; i<count; i++){
        pOut[i] = MAX(pIn1[i], pIn2[i]);
    }
}

L4:
    movdqu (%rsi,%rax), %xmm0
    addl $1, %r8d
    movdqu (%rdi,%rax), %xmm1
    pmaxub %xmm1, %xmm0
    movdqu %xmm0, (%rdx,%rax)
    addq $16, %rax
    cmpl %r8d, %r9d
    ja L4

L6:
    movzbl (%rdi,%rax), %edx
    movzbl (%r10,%rax), %esi
    cmpb %dl, %sil
    cmovae %esi, %edx
    movb %dl, (%r9,%rax)
    addq $1, %rax
    leal (%r8,%rax), %edx
    cmpl %edx, %ecx
    jg L6
void supLine(const PIXEL *pIn1,
    const PIXEL *pIn2,
    PIXEL *pOut, int count){
    for (int i=0; i< count; i++){
        pOut[i] =
            MAX(pIn1[i], pIn2[i]);
    }
}

L4:
    movdqu (%rsi,%rax), %xmm0
    addl $1, %r8d
    movdqu (%rdi,%rax), %xmm1
    pmaxub %xmm1, %xmm0
    movdqu %xmm0, (%rdx,%rax)
    addq $16, %rax
    cmpl %r8d, %r9d
    ja L4
void supLine(const PIXEL *pIn1,
    const PIXEL *pIn2,
    PIXEL *pOut, int count){
    for (int i=0; i< count; i++){
        pOut[i] =
            MAX(pIn1[i], pIn2[i]);
    }
}

L4:
    vmovdqu (%rsi,%rax), %xmm1
    addl $1, %r8d
    vmovdqu (%rdi,%rax), %xmm0
    vpmaxub %xmm0, %xmm1, %xmm0
    vmovdqu %xmm0, (%rdx,%rax)
    addq $16, %rax
    cmpl %r8d, %r9d
    ja L4
void supLine(const PIXEL *pIn1,
    const PIXEL *pIn2,
    PIXEL *pOut, int count){
  for (int i=0; i<count; i++){
    pOut[i] =
      MAX(pIn1[i], pIn2[i]);
  }
}

L16:
  vmovdqu (%rsi,%rax), %xmm1
  addl $1, %r8d
  vmovdqu (%rdi,%rax), %xmm0
  vinserti128 $0x1, 16(%rsi,%rax), %ymm1, %ymm1
  vinserti128 $0x1, 16(%rdi,%rax), %ymm0, %ymm0
  vpmaxub %ymm0, %ymm1, %ymm0
  vmovdqu %xmm0, (%rdx,%rax)
  vextracti128 $0x1, %ymm0, 16(%rdx,%rax)
  addq $32, %rax
  cmpl %r8d, %r9d
  ja L16
void supLine(const PIXEL *pIn1,  
    const PIXEL *pIn2,  
    PIXEL *pOut,    
    int count)    
{
    while(count--) {  
        *pOut++ =  
            MAX(*pIn1, *pIn2);    
        pIn1++;pIn2++;    
    }
}

L16:
  vmovdqu (%rsi,%rax), %xmm1
  addl $1, %r8d
  vmovdqu (%rdi,%rax), %xmm0
  vpcmpgtq %xmm0, %xmm1, %xmm2
  vbroadcast %xmm2, %xmm1, %xmm0, %xmm0
  vmovdqu %xmm0, (%rdx,%rax)
  addq $16, %rax
  cmpl %r8d, %r9d
  ja L16
int countDecr(int count){
    return count-1;
}
void supLine(const PIXEL *pIn1,
const PIXEL *pIn2,
PIXEL *pOut,
int count)
{
    while(count) {
        *pOut++ =
        MAX(*pIn1, *pIn2);
        pIn1++;pIn2++;
        count = countDecr(count);
    }
}

L17:
    movq (%r12,%rbx), %rcx
    movl %eax, %edi
    movq 0(%r13,%rbx), %r8
    cmpq %rcx, %r8
    cmovge %r8, %rcx
    movq %rcx, 0(%rbp,%rbx)
    addq $8, %rbx
    call __Z9countDecri
    testl %eax, %eax
    jne L17
1 Introduction

2 SMIL: General presentation

3 Optimizations

4 Benchmarks/Perspectives
SMIL: General

Base Objectives

- Speed
- Flexible licence
- Ease of integration:
  - Light
  - No major dependency
  - Multi-platforms/compilers

Additional Objectives

- Easy to use
- Easy to develop/extend
- A lot of factorization:
  - Less code to write
  - Facilitates major evolutions
  - Concentrate optimizations parts
SMIL: Ingredients

- C++
- Templates, but no real genericity (non-standard types require specializations)
- Auto-vectorization
- OpenMP
- CMake
- Swig
- (doxygen, git)
- BSD Licence
Intrinsic Specializations

```c
#ifdef __SSE_
    template<>
    inline void t_LineArithSup1D<UINT8>(...) {
        int i;
        __m64 r0,r1;
        for(i=0 ; i<=size ; i+=8)  {
            r0 = *((__m64 *) linein1);
            r1 = *((__m64 *) linein2);
            r1 = _mm_max_pu8(r0,r1);
            *(((__m64 *) lineout)) = r1;
            linein1 += 8;
            linein2 += 8;
            lineout += 8;
        }
    }
#else
    #ifdef __SSE2__
    template<>
    inline void t_LineArithSup1D<UINT8>(...) {
        int i;
        __m128i r0,r1;
        for(i=0 ; i<=size ; i+=16)  {
            r0 = _mm_load_si128((__m128i *) linein1);
            r1 = _mm_load_si128((__m128i *) linein2);
            r1 = _mm_max_epu8(r0,r1);
            _mm_store_si128((__m128i *) lineout,r1);
            linein1 += 16;
            linein2 += 16;
            lineout += 16;
        }
    }
#else...
#endif...
#endif...
#endif...
```

Matthieu Faessel, Michel Bilodeau

SMIL : Simple Morphological Image Library
Intro duction
SMIL: General presentation
Optimizations
Benchmarks/Persp ectives

Templates and Vectorization

Intrinsic Specializations

With Auto-Vectorization

Matthieu Faessel, Michel Bilodeau

SMIL : Simple Morphological Image Library
CMake

Cross-platform native makefiles and workspaces generator

- Allows to build the sources on almost any platform and with any compiler
- Facilitates cross-compilation
- Handles compilation options (external libs, optimizations, documentation, ...)
- Handles wrapped languages and types

Current tested platforms:

- Linux (32/64 bits) - GCC
- Windows (32/64 bits) - GCC, MSVC (without auto-vectorization)
- OSX - GCC, Clang
- Android
Current SMIL version allows to generate interfaces for **Java**, **Python**, **Octave** and **Perl**. Generated code have the same function names and arguments as original C++ code. The wrapped types and languages are defined via CMake.
Why Simple?

Easy to use

- Simple framework (only 2D/3D images, no complex structures due to genericity)
- $\sim$13,000 code lines (MorphM: $\sim$130,000)
- Multi-platforms, multi-compilers (CMake)
- Several possible languages/interpreters with common native C++ code and the same function names and arguments (swig): python, java, octave, perl, ...
- Integrated viewer, events management
- Simple and short function names
- Intuitive approach:
  - functions overload
  - operators overload
- webstart version, ...
Examples of use

```python
# Load an image
im1 = Image()
im1 << "lena.png"
im1.show()

# Equivalent to
read("lena.png", im1)

# Mask of values > 100
im2 = Image(im1)
im2 << ((im1>100) & im1)
im2.show()

# Equivalent to
grt(im1, 100, im2)
inf(im1, im2, im2)
```

Matthieu Faessel, Michel Bilodeau
SMIL : Simple Morphological Image Library
Examples of use

```python
# Sup of translations...
im3 = Image(im1)
sePts = ((0,0),(0,1),(1,1),(1,0),(-1,1),(-1,0),(-1,-1),(0,-1),(1,-1))
im3 <<= 0
for (dx,dy) in sePts:
    im3 |= trans(im1, dx, dy)
im3.show()
```

Matthieu Faessel, Michel Bilodeau
Examples of use

# Some thresholds
# Fixed threshold
threshold(im1, 100, 255, im2)
# Otsu (2 modalities)
threshold(im1, im2)
# Otsu (3 modalities)
otsuThreshold(im1, im3, 3)
im3.showLabel()
Why Simple?

**Effort to factorize as much as possible**

- line functors
- image functors (standard operations, morphological operations, ...)
- swig wrap definitions (simple swig macros to export functions/classes)
- ...

⇒ Most complex/redundant/optimization operations are handled jointly
⇒ Contributors do not need much knowledge to add new features

**Example: Dilation function**

```cpp
template<class t>
unaryMorphImageFunction<T, supLine<T>> dilateFunc();
```

⇒ SIMD, Parallelization, specialized SE functions, ...
Used optimizations

Software

- Line Processing
- Hierarchical Queues
- SE decomposition

Hardware

- SIMD
- Parallelization
- Binary Images
The classical approach: Neighborhood iterator

Pseudo-code (dilation):

```plaintext
for (j=0; j<height; j++)
    for (i=0; i<width; i++) {
        maxV = 0;
        for (it in sePts) {
            outPix(i,j) = MAX(maxV, inPix(i+it.x, j+it.y));
        }
        outPix(i,j) = maxV;
    }
```
The line approach: Using image translations

Pseudo-code (dilation):

```plaintext
for (j=0;j<height;j++) {
    fill(outLine(j),0);
    for (it in sePts) {
        tmpLine=translate(inLine(j+it.y),it.x);
        for (i=0;i<width;i++) {
            outLine(i)=MAX(outLine(i),tmpLine(i));
        }
    }
}
```
The line approach: Using image translations

Pseudo-code (dilation):

```c
for (j=0; j<height; j++) {
    ll(outLine(j),0);
    for (it in sePts) {
        tmpLine=translate(inLine(j+it.y),it.x);
        for (i=0; i<width; i++) {
            outLine(i)=MAX(outLine(i),tmpLine(i));
        }
    }
}
```
Line processing in morphological operations

Gains

- Speed (>x2)
- Allows some parallelization
- Allows to use SIMD
Example function: Morphological Dilation

Morphological dilation on 1024x1024 8bits image using a square structuring element:

Base implementation (neighbor iterator) 45 msecs
Line implementation 20 msecs (x2.25)
+ SE decomposition 7 msecs (x6.4)
+ SIMD 0.68 msecs (x66)
+ OpenMP 0.33 msecs (x136)

(Intel® Core™ i3 CPU M330 @2.13GHz, 2 cores, 4 threads)
1. Introduction

2. SMIL: General presentation

3. Optimizations

4. Benchmarks/Perspectives
**Benchmarks**

**Base Operations Process Time (msecs)**

*UINT8 Image (1024x1024)*

- **copy**: 0.02
- **fill**: 0.01
- **inv**: 0.02
- **add**: 0.04
- **sub**: 0.05
- **mul**: 0.59
- **div**: 0.96
- **inf**: 0.03
- **sup**: 0.03
- **equ**: 0.29
- **low**: 0.25
- **dilate**: 0.11
- **open**: 0.4

*Intel® Xeon® CPU E3-1245 @3,30GHz, 4 cores, 8 threads*

Matthieu Faessel, Michel Bilodeau

**SMIL**: Simple Morphological Image Library
EDIT 30/01/14: the FastMorphM functions used for this graph are ImDilateWithRect and ImOpenWithRect. Using the functions ImDilateWithSquare and ImOpenWithSquare, the actual results for FastMorphM are respectively 5.59 msecs and 11.81 msecs.

(Matplotlib) Xeon® CPU E31245 @3.30GHz, 4 cores, 8 threads)

Matthieu Faess, Michel Bilodeau | SMIL: Simple Morphological Image Library
Benchmarks

Speed Ratio Mono/Multi-Threads
(image 1024x1024, 8Bits)

(Intel® Core™ i3 CPU M330 @2.13GHz, 2 cores, 4 threads)

Matthieu Faessel, Michel Bilodeau

SMIL : Simple Morphological Image Library
Benchmarks

UIT8/BIT Execution time

Dilation with Hex SE (Image height: 1024)

- Blue line: UINT8 (msecs)
- Red line: BIT (msecs)

Image width (pix): 0 to 4500
Time (msecs): 0 to 0.4

Intel® Xeon® CPU E31245 @3,30GHz, 4 cores, 8 threads

Matthieu Faessel, Michel Bilodeau

SMIL: Simple Morphological Image Library
## SLOC

<table>
<thead>
<tr>
<th>SLOC</th>
<th>Directory</th>
<th>SLOC-by-Language (Sorted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3440</td>
<td>Morpho</td>
<td>cpp=3440</td>
</tr>
<tr>
<td>3107</td>
<td>Base</td>
<td>cpp=2941, python=166</td>
</tr>
<tr>
<td>2023</td>
<td>Core</td>
<td>cpp=2023</td>
</tr>
<tr>
<td>1551</td>
<td>Gui</td>
<td>cpp=1551</td>
</tr>
<tr>
<td>1519</td>
<td>NSTypes</td>
<td>cpp=1519</td>
</tr>
<tr>
<td>749</td>
<td>IO</td>
<td>cpp=749</td>
</tr>
<tr>
<td>430</td>
<td>doc</td>
<td>python=430</td>
</tr>
<tr>
<td>297</td>
<td>CMake</td>
<td>python=297</td>
</tr>
<tr>
<td>222</td>
<td>Addons</td>
<td>cpp=222</td>
</tr>
<tr>
<td>51</td>
<td>test</td>
<td>cpp=31, java=20</td>
</tr>
<tr>
<td>0</td>
<td>top_dir</td>
<td>(none)</td>
</tr>
</tbody>
</table>

Totals grouped by language (dominant language first):
- cpp: 12476 (93.18%)
- python: 893 (6.67%)
- java: 20 (0.15%)

Total Physical Source Lines of Code (SLOC) = 13,389
Development Effort Estimate, Person-Years (Person-Months) = 3.05
(Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months) = 0.82 (9.82)
(Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule) = 3.73
Total Estimated Cost to Develop = $411,839
(average salary = $56,286/year, overhead = 2.40).

SLOCCount, Copyright (C) 2001-2004 David A. Wheeler

Matthieu Faessel, Michel Bilodeau
Still much work to be done...

- More complex operations optimization:
  - Median, mean, ...
  - Labellization
  - ...

- Hierarchical queues optimization:
  - watershed
  - geodesic reconstruction

- GPU?
Thank you...

SMIL web page:

http://cmm.ensmp.fr/~faessel/smil/

Online WebStart Version (requires Java):