Applying Generic Programming to Image Processing

T. Géraud, Y. Fabre, A. Duret-Lutz

EPITA Research and Development Laboratory
http://www.lrde.epita.fr
Applying GP to IP

- Problem to solve
- Programming Paradigms
  - C-like
  - between C and C++
  - classic C++
  - generic C++
- Conclusion
Problem to solve

- A general-purpose library dedicated to IP.

- A lot of different applications in IP
  - many image types
    - 2D images, 3D images...
  - many data types
    - integers, floats, RGB...
  - many algorithms
    - e.g. add a constant value to pixel values...
Problem to solve

- Number of IP operators: $n_I \times n_D \times n_A$

- Implementing an algorithm ➔
  - $n_I \times n_D$ types to handle
    - i.e. $n_I \times n_D$ procedures to code !
  - most of libraries are restricted to very few input types!

- We want a single and efficient procedure per algorithm...
Programming paradigms

- Most libraries have a C-like style.
- Few libraries are (poorly) object-oriented.
- The GP paradigm is quite new.

Warning: you are going to see some code and for clarity and shortening purposes some syntactic sugar has been dissolved.
This 2D image structure is generic vis-à-vis the data type:

```c
struct Image2D
{
    int     nrows;
    int     ncols;
    type_t  datatype;
    void*   data;
};
```
C-like programming

This procedure seems to be generic vis-à-vis the data type:

```c
void add( Image2D* ima, double val ) {
    switch ( ima->datatype ) {
    case INT_U8:
        for (i = 0; i < ima->nrows; ++i)
            for (j = 0; j < ima->ncols; ++j)
                (ima->data)[i][j] += (val);
        break;
    case ...
    }
```
C-like programming

- Actually, we have \( n_I \times n_D \) procedures per algorithm.

- C-like paradigm leads to:
  - no code reusability,
  - tedious coding & maintaining processes,
  - limited libraries.
Between C and C++

This 2D image structure is also generic vis-à-vis the data type:

```cpp
template< typename T >
class Image2D
{
    int  nrows;
    int  ncols;
    T*   data;
};
```

Image2D<int_u8>
Image2D<double>
Image2D<bool>
Image2D<RGB>
But now this procedure is truly generic vis-à-vis the data type:

```cpp
template< typename T >
void add( Image2D<T>* ima, T val )
{
    for (i = 0; i < ima->nrows; ++i)
        for (j = 0; j < ima->ncols; ++j)
            ima->data[i][j] += val;
}
```
Between C and C++

- We have only $nl$ procedures per algorithm 😊

- but we are not yet generic vis-à-vis the image type... 😞
An image is an abstract type which declares an abstract method:

```cpp
template< typename T > class Image {
    Iterator<T>& create_i() = 0;
};
```

An iterator is used to browse data:

```cpp
template< typename T > class Iterator {
    void first() = 0;  T& data() = 0;  ... 
};
```
A 2D image is an image (inheritance):

```cpp
template< typename T >
class Image2D : public Image<T> {
    int  nrows;
    int  ncols;
    T*   data;
    Iterator2D<T>& create_i() { ... } 
};
```

and can create the proper iterator...
...which is a 2D iterator:

```cpp
template< typename T >
class Iterator2D : public Iterator<T> {
    int row;
    int col;
    Image2D<T>& ima;
    void first() { row = col = 0; }
    T& data() { return ima.data[row][col]; }
    ...
};
```
Last, this procedure is fully generic:

```cpp
template< typename T >
void add( Image<T>& ima, T val )
{
    Iterator<T>& i = ima.create_i();
    for (i.first(); i.is_ok(); i.next())
        i.data() += val;
}
```

but abstract method calls are prohibitive.
Classic C++

- We have solved the genericity problem 😊

- but we now have a performance problem! 😞
Generic C++

From the 2D image, we can deduce types:

```cpp
template< typename T >
class Image2D {
    int  nrows;
    int  ncols;
    T*   data;

typedef  Iterator2D<T>  iterator_t;
typedef T  data_t;
};
```
The procedure is parameterized by the input type \( I \) and type aliases are used:

```cpp
template< typename I >
void add( I& ima, I::data_t val )
{
    I::iterator_t i( ima );
    for (i.first(); i.is_ok(); i.next())
        i.data() += val;
}
```
At a procedure call, types are resolved by the compiler; so, we can have:

```cpp
template void add( Image2D<float>& ima, float val )
{
    Iterator2D<float> i( ima );
    for (i.first(); i.is_ok(); i.next())
        i.data() += val;
}
```
and this code is expanded by the compiler before the binary code is produced:

```cpp
void add( Image2D<float>& ima, float val )
{
    i.row = i.col = 0; /* i.first(); */
    while ( i.row < ima.nrows /* i.is_ok() */ )
    {
        ima.data[i.row][i.col] /* i.data() */ += val;
        /* i.next(); : */
        if (++i.col == ima.ncols) { i.col=0; ++i.row; }
    }
}
```
Generic C++

- We have fully generic procedures and
- they are efficient at run-time.

Moreover
- implementation is very close to algorithm description in natural language.
## Programming paradigms

<table>
<thead>
<tr>
<th></th>
<th>G./data</th>
<th>G./image</th>
<th>efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>--</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>C+</td>
<td>+</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td>C++</td>
<td>+</td>
<td>+</td>
<td>--</td>
</tr>
<tr>
<td>GP</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
Conclusion

- Generic Programming is a relevant paradigm for scientific computing such as image processing.

- This paradigm is not so easy to handle for complex algorithms.

- A first release at Dec. 2001: our IP library milena...