Building Software for Image Processing: A Generic Approach

Marcos César Cardoso Carrard
DeTec/Unijuí – Universidade Regional do Noroeste do RS
Caixa Postal 560, 98.700-000 Ijuí, RS, Brazil
carrard@unijui.tche.br

Marcos Cordeiro d'Ornellas
LACESM/CT/UFSM–Laboratório de Ciências Espaciais de Santa Maria
Av. Roraima s/n - Campus Universitário, 97105-900 Santa Maria, RS, Brazil
ornellas@inf.ufsm.br

Abstract

This paper presents a set of tools for image processing software development which aims to join both theory and practice into a trustworthy, efficient and generic solution that overlaps all data types and data structures. Indeed it concentrates on theoretical aspects and implementation mechanisms needed to apply a generic approach to image processing. Such tools are based on image algebra and complete lattices.

Keywords: image algebra, image processing, generic programming, complete lattices, software development, implementation techniques, and `C++` and `STL`.

1 Introduction

The need for code reuse is often an element of discussion in software engineering. However, there were a few situations where code reuse could be described in practical terms since its implementation lacked of tools and programming languages that might have fostered image processing software development. The C++ language and the Standard Template Library (STL) were the first to shed some light on algorithms that could be fully reused. By adding such a language with an extensive support given by image algebra and complete lattices, there is a chance for the development of software for image processing geared for several applications.

This paper concentrates on some of the technical issues associated with widespread software design when applied to image processing. It presents the generic programming paradigm by explaining how its tools can effectively be applied to solve some of the problems often found in the design and implementation of image processing operators and operations. It also shows how these operators can be used in the construction of software that is steady and uniform. The intention here is not only to cover all the aspects involved in the project of software for image processing, but also to focus in the general concepts involved in this process and in the necessary background that is needed to support them.

This paper is organized as follows. Section 2 presents the main characteristics of the software development for image processing and scientific applications. Then, section 3 shows a new paradigm of programming that allows for the development of reusable components. Sections 4 and 5 concentrate on the mathematical framework which is fully used in section 6 in order to establish the generic approach for image processing software construction. Finally conclusions are drawn in section 7.

2 Programming in Image Processing

Among programming paradigms, there were several initiatives were software developers aimed to organize and establish the real limits and the elements involved in the task of programming. This situation was more present in the field of scientific programming. The first mention in the literature about this subject can be found in [20] where a separation in terms of semantics was considered. This approach deals with algorithms and data structures separately and shows that an algorithm can be seen as a combination of them. With the evolution of programming techniques and the complexity of programming tasks, another important element was involved in this relation: data types to be manipulated by the algorithms and stored in data structures. Consequently, data types had evolved as a factor of dependence in the application development.

This relation can be better represented in the orthogonal form with a three-dimensional system of coordinates, where the axes represent data types (X), data structures (Y), and the algorithms (Z), respectively. Using a symbolic representation of programming activity, combinations among the three elements covering all possible cases have a clear exponential growth. By considering the classic nomenclature of the Cartesian system, the possibilities are equal to $X \times Y \times Z$.

Programming paradigms that shown up in the evolution of this activity concentrated their performance in just a parcel of this representation. For example, structured programming centered its action in the algorithms and data types, while the object oriented programming did the same in relation to the data structures and data types. Currently, the state of the art in terms of programming leads to a generic programming paradigm ([7, 17, 14, 13]), which is an alternative for the description of programs that work generically i.e., the structures and data types involved in this operation are distinct. In this way, algorithm representation fall to $X + Y + Z$.

The development of software for image processing is known to be a difficult matter. The reasons for this are the large and complex data, and specific solutions described for the problems due to the necessity of computational efficiency for several applications. From one side, it is easy to deal with local attributes of an image, like pixels brightness, neighborhood operations, edges and so forth. On the other side, there is a great complexity in dealing with the global content the image, mainly under a semantic point of view [16]. As a consequence, the description of programs and algorithms highly depends on the application context and on the image types to be considered.

Based on the development of a typical image processing solution, it is possible to perceive that an extensive group of image applications were developed within typical and traditional computer programming paradigms. Therefore, they became dependent on internal details like data types, for instance. A direct consequence of this is the need of rewriting code for each new situation. In this way code grows faster which is difficult to support and maintain.

2.1 The Problem

One of the problem statements addressed in this paper was first mentioned in [7] including an evaluation of software packages and libraries with respect to mathematical morphology in terms of implementation aspects, utilization and documentation, operation set, and so forth. It has been shown that the implementation task can also be extended for general image processing applications. Indeed, it also presented that the overall alternatives for the development

of image processing operators do not reduce the complexity in terms of code, that is, code needs to be written for every particular situation, which goes against the generic programming paradigm [14].

A generic approach for image processing software development presented in this paper wants to answer some essential questions with respect to theoretical and practical aspects involved during implementation. These questions are listed below:

- Which are the mechanisms involved in order to build software for image processing that assures algorithm generality?
- Which would be the appropriate design tools to be used to implement solutions for the recurring problems in the area?
- How to describe algorithmic solutions only once and, in addition, to assure that these solutions would be reused in other situations?
- Which are the complexity limits in this approach as to describe efficient algorithms?

The answers for these questions will be given along the paper.

3 Generic Programming

The generic programming area is becoming attractive among software development paradigms. In algorithm specification for any data structure, the notion of generality appears in terms of code reuse for many contexts. Even though generic programming is not quite a new concept, the idea has attracted a lot of programmers and researchers within the software engineering community [8, 15]. The reason for this is largely based on the advent of tools and mechanisms within the programming languages that allow the implementation of generic concepts and on the reduction of programming efforts needed for the task in hand.

It should be noted that the generic programming approach tries to combine a set of well-known advantages from traditional programming paradigms, which were set aside in the past, with generic implementation principles. For instance, the abstraction, reuse and adaptation capabilities in the object-oriented paradigm are not disrespected but added to a generic and functional implementation. Thus, recurring problems like the use of virtual functions and inheritance are minimized. This also happens with environment and type dependencies that are present while using pointers. With respect to structured program paradigm, the proximity with generic programming is not so clear since data abstraction tends to hide the complexity of the implementation.

According to [7], the goal of object oriented paradigm is to specify interfaces capable to encapsulate data structures and algorithms. After a class is constructed, the implementation details are said of little importance. In this way, a reusability problem shows up because object oriented programming has a strong reliance in regards to data relationships and besides it does not consider all the important aspects of correctness and efficiency which are difficult to evaluate due to the abstraction imposed. Algorithm analysis states that such an action can only be carried through when there is a full knowledge of the algorithm [1, 4]. This possibility is inversely proportional to the used level of abstraction in the code. Therefore, generic programming is an alternative for this problem. The same abstraction used for object oriented programming with respect to an object concept can also be applied for algorithm development within generic programming. In order to do so, it offers a diverse set of data structures allowing high flexibility in terms of choice and usage, as well as facilitates the development of new structures even when a native paradigm does not obtain a correct solution for a given problem. Thus, data structures are managed like data deposits that are used and manipulated by the algorithms, which tend to be abstract since they look forward to independence of the data types that needs to be manipulated.

Regarding the context of reusability of functional characteristics in some programming paradigms, generic programming is enclosed by means of extensions and libraries that are attached to code generators. Examples of such languages are `Ada`, `Modula-3`, `Fortran`, and `C++`. It is important to stress the fact that the programming paradigm is addressed within the `C++` language in terms of the Standard Template Library as known as `STL`. This library allows for a generic programming approach for many applications and environments. Although developed using an object oriented language based on templates, `STL` uses a flexible mechanism that employs generic algorithms into a variety of data structures. `C++` flexibility in conjunction with the best of object oriented and generic paradigms results in a highly extensible library that is rarely found in development environments.

Traditional applications in image processing are tied together with their goals and with the environment where they were implemented. Besides, there is an implicit complexity in this process in regard to the semantic interpretation of the image based on its content. In order to reduce such a complexity, more and more specific applications are developed. Generic programming can be used in the image processing software developments in many ways. Firstly, an image has a particular nature, which needs to be represented according to the available data structures. In this way, the data structures definition allows for image representation and manipulation with many different

data types like integers, floats, and so forth. Secondly, generic programming strive for algorithm development that manipulates data structures which express the computational logic without giving so much emphasis to the image semantics.

However, even though efficiency problems related to object oriented programming are set aside, a generic programming paradigm is not enough to assure the development of efficient programs. Therefore, the design of data structures and algorithms for image processing must aim at producing well-posed solutions even when they describe a set of mathematical definitions that have a strong background like image algebra. Well-posed solutions are those that produce correct results while using little computational resources. These resources can vary according to the problem but typically are measured in terms of time or memory usage.

3.1 C++ and STL

One of the most popular mechanisms in the object oriented languages used to guarantee a high degree of abstraction with respect to the application as well as code reusability are grounded in some concepts like inheritance, virtual functions, and polymorphism. Within C++ language these characteristics are known to be the most important features for the object oriented paradigm. On the other side, it can be shown that such programming resources, mainly inheritance and virtual functions, aggregate a highly computational cost based on the available resources used to implement these features in terms of the execution time (e.g. dynamic binding) [5, 6].

C++ brings us with another susceptible mechanism to the generation of abstract and reused code without losing the natural efficiency often found in this language. This is the `template` resource that describes algorithms in a logical structure with independence of data types to be manipulated. In the algorithm 1, the `findA` shows, for instance, the search of a integer value in a vector of integers, while the function `findB` in algorithm 2 makes the same but with independence of data types. To make this clear, the second function uses a template mechanism that establishes a generic data type `T` to be manipulated by the code. Thus, the template mechanism deals with any data types in the language and even to newly built types since a set of logical operators and operations is supported. The efficiency of this implementation is likely to be observed in compile time because this function is automatically rewritten for each data type used in the program, in a way that seems transparent for the programmer.

```
inline int *findA( int *first, int *last, int value )
{
    while( first != last && *first != value )
        first++;
    return first;
}
```

Algorithm 1: Implementation of a search algorithm with pointers.

```
template< class T >
inline T *findB( T *first, T *first, const T value )
{
    while( first != last && *first != value )
        first++;
    return first;
}
```

Algorithm 2: Implementation of a search algorithm with templates.

Although the templates mechanism supports generality to the available data types used in the program, this type of implementation still presents some difficulties to add the same generality for the data structures presented in the language. This functionality can be obtained with the Standard Template Library (STL) [14, 17, 18, 6].

STL is a development tool that, beyond the easiness of maintenance and code understanding, adds to the C++ language that supports a reusable component development approach. STL is structured in terms of `containers`, `iterators` and `algorithms`. Containers are data repositories (storage structures), while iterators are used to manipulate its contents by means of algorithms. Although having some definition similarities and functionality with C++ pointers, iterators are not pointers because they does not have any relationship with the data type addressed. This guarantees the possibility for the development of generic algorithms for these data. Moreover, this proposal prevents the use of inheritance and virtual functions in favor of a generic programming based on templates [17]. This aims at a better execution performance without loss of generality. Although STL is often seen

as a library of containers, it is more appropriate to perceive it as a collection of generic algorithms combined with all the functionality required by these algorithms [2].

When considering a common search, STL includes the functionality for this task. Moreover, one can implement additional functions that make use of the data included in the container by only knowing a reference to this data. For instance, a `vector` container V contains a set of data (the type is irrelevant from an implementation point of view). The search algorithm could be described with a parameter pointing to the initial point (V.begin()) to the searched and another one to the end point (V.end()). Iterators for this container implement advanced mechanisms that allow for the navigation within the container. Even the type of search can also be generalized by means of a function object that is used to execute some kind of test that will influence in the stop condition e.g. it can find a first value which is less than a supplied value.

When an algorithm implemented in STL is described, it needs to be adjusted to the rules and statements of the library as way to comply with all available requirements. A common problem related with this comes from concepts that are not part of the programming language. They are expressed in the programming behavior and code documentation. Thus, the programmer must clearly establish the point in which this code is inserted in the hierarchy, which elements are supported, and which are the running conditions. Moreover, the use of a large documentation within the source is an important element for the software maintenance.

3.2 STL with Images

The application of STL in the field of image processing can be found in [11, 12]. This approach emphasizes the need for a container structure to store a basic image type. The specific types to be manipulated can inherit this container. Inheritance shares common characteristics related to stored data but does not offer a common interface for these data by means of virtual functions. If one considers that STL is organized in a hierarchical form, then image containers can use all the characteristics of a `vector container` with some relative restrictions to iterators to be used in the data access. In order to fully explore the image container, it is necessary to implement iterators that make navigation in the direction of the four neighbors of each pixel [16]. The need for a diagonal navigation can be supplied by a combination of these iterators. Iterator implementation is based on increment and decrement in regards to base iterators of a `vector` container. This allows for random access to any part of the image. In these conditions, the image is considered as a two-dimensional structure. Note, however that this framework can be surpassed for a larger data set representation like three-dimensional ones.

```
template< class ImageIterator >
void copy( ImageIterator f, ImageIterator d )
{
    IteratorImagem aux;
    IteratorImagem dest = d.begin();

    for( aux = f.begin(); aux != f.end(); )
        *dest++ = *aux++;
}
```

Algorithm 3: Generic copy of an image.

Algorithm 3 shows an example that describes an algorithm that makes a generic copy of an image. This functionality would also be implemented through operator overloading, but such an algorithm better characterizes the use of the STL in the context of images. Two images are used as parameters through its iterators f and d in the algorithm. Both images have the same image type. Based on these requirements, it is possible to describe a loop from the beginning (`f.begin()`) to the end of the image (`f.end()`). This loop uses one iterator to be incremented over de first image. There is a second iterator that points to the second image (result). This example requires that iterator operations must be coded properly, considering the necessary restrictions regarding image limits. It is important to stress that for every pixel in the original image, there is a value associated with the resulting image pixel and both iterators need to be incremented.

Functions objects can allow for the implementation of a more generic algorithm, mainly when the algorithm depends on some data or a specific structure. As such, in a convolution operation, the convolution mask can be applied in a separate procedure and the procedure itself be passed as parameter. This algorithm would be very similar to the one of the algorithm 3 by only changing the resulted attribution which is based on the object function involved.

Finally, it is important to emphasize that an implementation including these characteristics needs to grant correctness and efficiency of code only once. For every new image type considered in the program, the code

functionality is replicated automatically.

4 Image Algebra

Image Algebra is a mathematical framework for image processing and analysis [19]. It is based on the understanding that operators and operations on images are appropriately represented by mathematical entities. Therefore, any operation on images can be mathematically defined if it produces a correct result. This is strengthened by the fact that image processing and computer vision are grounded on a solid mathematical basis. In addition, classical literature about these two subjects makes clear that a mathematical framework is always needed in order to understand and implement the operators and operations on images as well as their implementation aspects. Above all, image algebra tries to unify image processing concepts by means of a set of mathematical definitions in a way to uniformly represent techniques and available solutions for image processing and computer vision problems.

Image algebra is termed a heterogeneous algebra, working with multiple sets of operations and operators on images. Since image processing involves not only operations on images but also different types and quantities associated with images, it seems natural that a set of operations and operators on images can be used to work on a varied collection of image types. An image needs to be considered as a collection of points in space where every point has an associated value [16]. Consequently, the definition of image algebra embraces two mathematical theories involving set theory, which works on a set of points and a valued set algebra that deals with values that are associated with these points.

Image algebra has a unique representation. However, there are many restrictions with respect to valued sets when they are used as basic elements for image representation. When a valued set is employed, it forces the definition of conjugated and complimentary operations that aim at some kind of uniformity in image processing. Furthermore, it tries to address a large collection of image types where each possible numerical set has its proper representations and operations as well as a way to use them.

This becomes a real problem for the use of image algebra as a pattern for generic software development in image processing. Image algebra is a heterogeneous algebra in its definition. Thus it seems unusual to apply it for valued sets which encloses an homogeneous algebra formed by a set of values in conjunction with a finite set of operations [19].

5 Complete Lattices

Complete lattice theory can be interpreted as a general framework for mathematical morphology [10], however its concepts can be applied for image processing as well. It was shown in [7] that this theory not only suits for non-linear image processing e.g. mathematical morphology, but also for linear image processing like shift-invariant operators and region growing techniques to name a few.

A partially ordered set (`poset`), is a nonempty set $\mathcal{L}$ together with an ordering relation $\leq$ that is reflexive, anti-symmetric and transitive [10]. A poset V is said to be a complete lattice when each subgroup $K \subseteq V$ has an infimum and supremum. A complete lattice representation is given by $\mathcal{L} = (V, \leq)$, where infimum and supremum are represented by $\wedge$ and $\vee$ respectively. The ordering relation $\leq$, is essential in a complete lattice. Regarding the uniformity of representation, infimum and supremum of a set V exist and are denoted by $\wedge V = -\infty$ and $\vee V = +\infty$ respectively.

An operator on a complete lattice $\mathcal{L}$ is a mapping $\phi : \mathcal{L} \to \mathcal{L}$. It should be noted that the set of all the operators in $\mathcal{L}$ with an induced order $\phi \leq \psi \Leftrightarrow \forall v \in \mathcal{L} : \phi(v) \leq \psi(v)$, is a complete lattice as well [10, 7]. This guarantees that the induced order on the operators keep the uniformity of representation.

If one considers an image as being a function $f : E \to V$ that maps a visual field E in a set of possible values V, the complete lattice theory can be applied to an image since the ordering relation $\leq$ is maintained. In view of this and for the sake of representation, the characterization proposed in [7] is used. Such a representation deals with two lattices: the pixel lattice and the image lattice. The former aims at representing those operations in the domain of pixel values or with local image attributes. The latter deals with operations on images in a general fashion.

A complete lattice can also be used in the context of a generic framework on images. For this reason, it is necessary to define an ordering relation for every possible set V, leading to an increased complexity. This that can be summarized by a simple example: How to order images in a image lattice or an RGB image in a pixel lattice? This complexity is due to the fact that the order of non-scalar values is not a natural process. The literature shows a large variety of alternatives to order non-scalar data [3, 7]. These alternatives present advantages as well as disadvantages that depend on the application and the data types. For instance, mathematical morphology when applied on color images can result in data, which is out of the scope of the original image. On the other hand, the application of a linear spread function in the same data does not produce the same effect.

The need for an ordering relation to be kept while complete lattice theory is used for the development of

software for image processing applications brings an important question with respect to efficiency. It is clear that the increase of image dimensionality and the non-scalar nature of pixels will have an impact on the computational complexity of the algorithms that keep the ordering relation, i.e. algorithms that respect the complete lattice. Two possible solutions to tackle this problem are addressed in this paper: one explores image representation structures that guarantee efficiency even with tricky situations, which can be addressed with generic programming and STL. Another solution pays attention to the minimization of image representation aiming at reducing the complexity growth. Even though it could appear as a restriction for the approach set forth in this paper, it is important to note that the great majority of image processing happens in the two-dimensional space. This space presents an acceptable level of efficiency for lattice usage. However, problems may arise when data dimensionality becomes larger. In such cases, an image decomposition seems to be the best solution as proposed in [9]. It should be noted that, even when the decomposition is not available, complete lattice theory keeps its uniformity.

6 A Generic Approach for Image Processing Software Development

6.1 Requirements

As already mentioned in this paper, image processing applications present a high degree of dependence in terms of the implementation goals and environment. It should be noted that a large set of image processing operators share a significant parcel of code. This is even clearer when identical or equivalent operators are applied for different image types. Usually, programmers have a tendency to rewrite code for every single type. This analysis allows us to deduce that there exists a concrete possibility to work with patterns for the development of image processing algorithms.

The construction of image processing operations and operators can be of high complexity. In the context of this paper, such complexity is accentuated since the goal is to develop software capable to deal with a diverse set of image types. Thus, the programmer does not have to be forced to write his own loops over all pixels in the, which is the fact of major dependence in image processing algorithms. In addition, software development must be structured in a concise form, allowing and giving guarantees that all requirements will be fulfilled. A list of basic requirements is given as follows:

- Correctness: the description of algorithmic solutions must have a solid theoretical background. The application of this requirement allows for the use of mathematical tools that gives evidence and guarantees the correctness of the developed application;

- Generality: a solution must be generic enough to overlap the particularities included in images and applications on images. This can be obtained through a higher degree of abstraction with respect to the data to be manipulated; the generic description of the patterns and algorithm behavior; and, with mechanisms for interchange and treatment of these data among application layers;

- Robustness: in image processing the majority of high-level decisions (application level) are based on syntax and predicates on the low-level functionality (processing level). Thus, the behavior of any software tool that manipulates images must be steady in all levels;

- Flexibility: An image processing tool must be adapted to the most varied conditions without losing its functionality. Moreover, it must be fully portable and steady in any environment;

- Efficiency: algorithms must be efficient, in time and space, to produce the desired result in terms of feasible conditions.

6.2 A Generic Approach

The main problem addressed in this paper is the lack of a general structure for the development of software for image processing. This section gives three propositions, which constitutes the generic approach needed to solve the problem. Propositions are given below:

1. Image algebra as a framework for operations and operators on images: The image algebra establishes a practical representation of a theory that is understandable, and singular in general image processing applications. Using it as a design pattern for image processing applications allows for a minimum set of operators and operations capable of representing the necessary functionality for the developed applications. It also makes clear that the programmer can use a great deal of typical mathematical structures and theorems of abstract algebra as guarantee for the implementation.

2. Complete lattices as an element for image representation: The use of valued sets as a fundamental concept in image algebra removes the uniformity of this representation. Therefore, a general structure like in [7] must be used, that is, a pixel lattice in conjunction with an image lattice. These definitions are more appropriate to the process of software development since they are much more in tune with the implementation levels. Thus, pixel lattices ($\mathcal{P}$) and image lattices ($\mathcal{I}$) are represented respectively by $\mathcal{P} = \{\mathcal{V}, \leq, \wedge, \vee, +\infty, -\infty\}$ and $\mathcal{I} = \{\mathcal{F}(\mathcal{E}, \mathcal{V}), \bigwedge_{\leq}, \bigwedge_{\wedge}, \bigwedge_{\vee}, \bigwedge_{+\infty}, \bigwedge_{-\infty}\}$. It should be noted that an image is characterized as a function F. Operators like `infimum` and `supremum`, as well as the limits of the lattice, are established around an ordering relation to be defined on images;

3. Software implementation by means of `C++` and `STL`: `STL` allows for the implementation of image algebra operations and operators in a generic fashion. It is possible to use this implementation in a great variety of image types minimizing code proliferation. Moreover, algorithms will be always correct and within efficiency limits. In addition, `STL` will propagate these guarantees for the future developments. Finally, a software implementation like this can be used with applications, libraries, and interfaces developed in languages such as `C`, `Java`, and `Python`. The same holds for its integration with programming environments like `Khoros`, `MatLab`, and `IDL`.

6.3 The Software Framework

The internal organization of a software framework for image processing foresees its construction in three distinct operational levels, called layers. The kernel gives support to libraries, interfaces, and application development functionalities.

The inner layer of the kernel, called base, aims at the definition of data structures images e.g. containers, its storage and recovery mechanisms, as well as all consistency guarantees that might be needed. In this layer, a generic implementation is necessary mainly in the interface with the next layer to allow a high degree of data abstraction. The communication with the manipulation layer is carried through a generic image class that implements all the functionality related to generic structure, storage and consistencies. The image class considers the image itself as complete lattice and defines storage mechanisms for all available valued sets V. This layer makes use of a `C++` language and `STL` to get these functionalities. The next goal is to implement and supply the intermediate layer with a set of elementary operations that fully express the great number of algorithms presents in low-level image processing. This layer is structured on top of image algebra theories. Finally, the upper layer is constructed as an interface between the elementary operations in the previous level with all possible developments that can be carried through around the kernel. With this software framework, a new library can be easily defined. It also needs to be adaptable and portable to the varied environments involved.

7 Conclusion

The main goal of generic programming paradigm is to build software with a larger degree of applicability that cannot be found among conventional paradigms. It aims at developing algorithms that are capable of overlap the specific characteristics in the development process such as data types and data structures while focusing in code reusability. Within this context, implementation complexity is always increased. Design and implementation implies in an additional effort to produce reused software even when compared with other traditional areas. This enforces the practical necessity of implementing all operations efficiently to deal with a large amount of data types. To address this problem, the use of generic programming in software development is essential as a practical element of reuse and continuity.

Moreover, this topic brings to discussion the need of finding algorithms and programs capable of represent correct and efficient solutions for particular situations in terms of image types. The goal is not only to prove the existence of these algorithms, but also to give guarantees of correction and provide for a clear interface mechanism.

Finally, this paper considers the software construction of a generic framework with an innovative development approach. Such framework includes one of the current trends in software engineering namely software patterns that will be fully used in the design and implementation steps.

References

[1] Alfred Aho and Jeffrey Ullman. *Foundations of Computer Science*. Computer Science Press, 1992.

[2] Matthew Austern. The SGI standard template library. *Dr. Dobb's Journal*, 268:18–27, 1996.

[3] V. Barnett. The ordering of multivariate data. *Journal of the Royal Statistcal Society*, 139:318–355, 1976.

[4] Thomas Cormer, Charles Leiserson, and Ronal Rivest. *Introduction to Algorithms*. McGraw-Hill, 1991.

[5] Jérôme Darbon, Thierry Géraud, and Alexandre Duret-Lutz. Generic implementation of morphological image operators. In *6th International Symposium on Mathematical Morphology - ISMM'2002*, April 2002.

[6] Harvey Deitel and Paul Deitel. *C++ Como Programar*. Ed. Bookman, 3^a edition, 2001.

[7] Marcos Cordeiro d'Ornellas. *Algorithmic Patterns for Morphological Image Processing*. PhD thesis, University of Amsterdam, 2001.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[9] Robert Haralick and Linda Shapiro. *Computer and Robot Vision*, volume I. Addison-Wesley, 1993.

[10] Henk Heijmans. *Morphological Image Operators*. Academic Press, 1994.

[11] Ullrich Köthe. Reusable software in computer vision. In B. Jähne, H. Haussecker, and P. Geissler, editors, *Handbook of Computer Vision and Applications*, volume 3, pages 103–132. Academic Press, 1999.

[12] Ullrich Köthe. STL - style generic programming with images. *C++ Report Magazine*, 12(1):24–30, January 2000.

[13] Didier Parigot Loïc Correnson, Etienne Duris and Gilles Roussell. Generic programming by program composition. In *Workshop on Generic Programming*, Swedwn, 1998.

[14] David Musser and Alexander Stepanov. Algorithm-oriented generic libraries. *Software - Pratice and Experience*, 24(7):623–642, 1994.

[15] Dirk Riehle. Composite design patterns. In *Object-Oriented Programming Systems, Language and Applications - OOPSLA*, pages 218–228, 1997.

[16] Milan Sonka, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision*. Brooks/Cole Pub., 2nd edition, 1999.

[17] Alexander Stepanov and Meng Lee. The standard template library. Technical Report 94-34, Hewlett-Packard Laboratories, 1995.

[18] Johannes Weidel. The standard template library tutorial. http://www.eecs.lehigh.edu/resources/STL-tut/prwmain.htm (April/2002), April 1996.

[19] Joseph Wilson and Gerhard Ritter. *Handbook of Computer Vision Algorithms in Image Algebra*. CRC Press, 2nd edition, 2000.

[20] Nicholas Wirth. *Algorithms + Data Structures = Programs*. Prentice-Hall, 1976.