From Digital Images to Simplicial Complexes: A Report*

ForMath La Rioja node

Departamento de Matemáticas y Computación, Universidad de La Rioja,
Edificio Vives, Luis de Ulloa s/n, E-26004 Logroño (La Rioja, Spain).

Abstract. A methodology to study digital images by means of simplicial complexes, a basic notion in Algebraic Topology, is presented in this report. Moreover, a formalization in both the ACL2 Theorem Prover and the proof assistant Coq of the correctness of our algorithms is given.

1 Introduction

Algebraic Topology is a complex and abstract mathematical subject; however, some of its techniques can be applied to different contexts such as coding theory [15], data analysis [8], robotics [10] or digital image analysis [6,7] (in this last case, in particular in the study of medical images [14]).

Here, we are going to focus on the application of Algebraic Topology to the study of binary digital images. In the Algebraic Topology framework, binary digital images can be studied using simplicial complexes. This section is devoted to present a technique based on simplicial complexes to study binary digital images. In particular, we study monochromatic (black and white) digital images by means of algorithms related to simplicial complexes.

It is worth noting that if we want to apply our method in real life problems (for instance, in the study of medical images), we must be completely sure that the results produced by our programs are correct. Therefore, the formal verification of our programs with a Theorem Prover tool is significant. In particular, we have used both the ACL2 Theorem Prover [9] and the proof assistant Coq [4,2] as well as the SSReflect extension [5] and the libraries it provides.

The rest of this report is organized as follows. Section 2 introduces the basic background about simplicial complexes and some algorithms about them. Section 3 presents the technique and the algorithms that we apply to analyze digital images by means of simplicial complexes. Remarks about the formalizations of our method in ACL2 and Coq/SSReflect are provided respectively in sections 4 and 5. This report ends with a section of Conclusions and Further work.

* Partially supported by Ministerio de Ciencia e Innovación, project MTM2009-13842-C02-01, and by European Community FP7, STREP project ForMath.

2 Mathematical Preliminaries

In this section, we briefly provide the minimal mathematical background needed in the rest of the paper. We mainly focus on definitions. Many good textbooks are available for these definitions and results about them, the main one being maybe [11].

Let us start with the basic terminology. Let V be an ordered set, called the *vertex set*. An *(ordered abstract) simplex* over V is any ordered finite subset of V. An *(ordered abstract) n-simplex* over V is a simplex over V whose cardinality is equal to $n+1$. Given a simplex α over V, we call *faces* of α to all the subsets of α.

Definition 1 An *(ordered abstract) simplicial complex* over V is a set of simplexes $\mathcal{K}$ over V such that it is closed by taking faces (subsets); that is to say:

$$\forall \alpha \in \mathcal{K},\ if\ \beta \subseteq \alpha \Rightarrow \beta \in \mathcal{K}$$

Let $\mathcal{K}$ be a simplicial complex. Then the set $S_n(\mathcal{K})$ of n-simplexes of $\mathcal{K}$ is the set made of the simplexes of cardinality $n+1$ of $\mathcal{K}$.

Example 2 Let us consider $V = (0, 1, 2, 3, 4, 5, 6)$.

The small simplicial complex drawn in Figure 1 is mathematically defined as the object:

$$\mathcal{K} = \left\{ \begin{array}{l} \emptyset, (0), (1), (2), (3), (4), (5), (6), \\ (0,1), (0,2), (0,3), (1,2), (1,3), (2,3), (3,4), (4,5), (4,6), (5,6), \\ (0,1,2), (4,5,6) \end{array} \right\}.$$

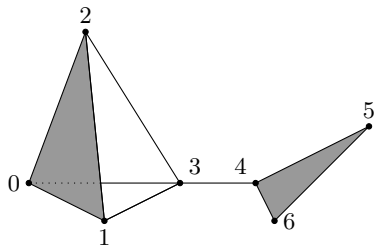

Fig. 1. Butterfly Simplicial Complex

Note that, because the vertex set is ordered the list of vertices of a simplex is also ordered, which allows us to use a sequence notation $(\ldots)$ and not a subset notation $\{\ldots\}$ for a simplex and also for the vertex set V. It is also worth noting that simplicial complexes can be infinite. For instance if $V = \mathbb{N}$ and the simplicial complex $\mathcal{K}$ is $\{(n)\}_{n\in\mathbb{N}} \cup \{(n-1, n)\}_{n\geq 1}$, the simplicial complex obtained can be seen as an infinite bunch of segments.

Definition 3 A *facet* of a simplicial complex $\mathcal{K}$ over V is a maximal simplex with respect to the subset relation, $\subseteq$, among the simplexes of $\mathcal{K}$.

Example 4 The facets of the small simplicial complex depicted in Figure 1 are:

$$\{(0,3),(1,3),(2,3),(3,4),(0,1,2),(4,5,6)\}$$

Let us note that a *finite* simplicial complex can be generated from its facets taking the set union of the power set of each one of their facets. In general, we have the following definition.

Definition 5 Let $\mathcal{S}$ be a finite sequence of simplexes, then the set union of the power set of each one of the elements of $\mathcal{S}$ is, trivially, a simplicial complex called the *simplicial complex associated with* $\mathcal{S}$.

It is worth noting that the same simplicial complex can be generated from two different sequences of simplexes; in addition, the minimal sequence of simplexes which generates a finite simplicial complex is the sequence of its facets.

Then, the following algorithm can be defined.

Algorithm 6
Input: a sequence of simplexes $\mathcal{S}$.
Output: the associated simplicial complex with $\mathcal{S}$.

The correctness of this algorithm was verified both in ACL2 and Coq [3].

3 The framework to study digital images

Let n be any positive integer. An *n-xel* q in an Euclidean n-space, $\mathbb{R}^n$, is a closed unit n-dimensional (hyper)cube $q \subset \mathbb{R}^n$ whose 2^n vertices have natural coordinates (more precisely, an *n-xel* in $\mathbb{R}^n$ is a cartesian product like $[i_1, i_1 + 1] \times [i_2, i_2 + 1] \times \ldots \times [i_n, i_n + 1]$). In this memoir, a *pixel* is a 2-xel in $\mathbb{R}^2$. We define an *n-dimensional binary image* or *nD-image*, to be a finite set of n-xels in $\mathbb{R}^n$.

An nD-image $\mathcal{I}$ can, of course, be represented by a finite n-dimensional array of 1's and 0's in which each 1 represents an n-xel in $\mathcal{D}$ and each 0 represents an n-xel that is not in $\mathcal{D}$. Let us focus on the study of nD-images by means of simplicial complexes. Firstly, we present the study for the cases of 2D-images and eventually the general case.

As we have just said, a 2D-image $\mathcal{D}$ can be represented by a finite 2-dimensional array of 1's and 0's in which each 1 represents a pixel in $\mathcal{D}$ and each 0 represents a pixel that is not in $\mathcal{D}$ (in a monochromatic 2D-image $\mathcal{D}$, black pixels are represented by 1's, on the contrary white pixels are represented by 0's).

Let $\mathcal{D}$ be a 2D-image codified as a 2-dimensional array of 1's and 0's. We want to associate a simplicial complex with $\mathcal{D}$. From a digital image, there are several ways of constructing a simplicial complex (see [1]). The approach

that we have followed here consists of obtaining from $\mathcal{D}$ the facets of one of its associated simplicial complexes. Subsequently, applying Algorithm 6 (the algorithm which constructs a simplicial complex from a sequence of simplexes), we obtain a simplicial complex associated with $\mathcal{D}$.

The process that we have followed to obtain the facets from a 2D-image $\mathcal{D}$ is as follows. Let $V = (\mathbb{N}, \mathbb{N})$ be the vertex set, that is, a vertex, in this case, is a pair of natural numbers. Let $p = (a, b)$ be the coordinates of a pixel in $\mathcal{D}$ (that is, the position of the pixel in the 2-dimensional array associated with $\mathcal{D}$). From p we can obtain two 2-simplexes that are two facets of the simplicial complex associated with $\mathcal{D}$. Namely, from $p = (a, b)$ we obtain the following facets: the triangles $((a, b), (a+1, b), (a+1, b+1))$ and $((a, b), (a, b+1), (a+1, b+1))$. If we repeat the process for the coordinates of all the pixels in $\mathcal{D}$, we obtain the facets of a simplicial complex associated with $\mathcal{D}$, that will be denoted by $\mathcal{K}_{2D}(\mathcal{D})$.

Therefore, we can define the following algorithm.

Algorithm 7
Input: a 2D-image $\mathcal{D}$ represented by means of a 2-dimensional array of 1's and 0's.
Output: the facets of $\mathcal{K}_{2D}(\mathcal{D})$, a simplicial complex associated with $\mathcal{D}$.

Example 8 Consider the 2D-image depicted in the left side of Figure 2. This image can be codified by means of the 2-dimensional array: $((1, 0), (0, 1))$, then, the coordinates of the black pixels are $(0, 0)$ and $(1, 1)$. Therefore, applying Algorithm 7 we obtain the facets of $\mathcal{K}_{2D}(\mathcal{D})$:

$$(((0, 0), (0, 1), (1, 1)), ((0, 0), (1, 0), (1, 1)), ((1, 1), (1, 2), (2, 2)), ((1, 1), (2, 1), (2, 2))).$$

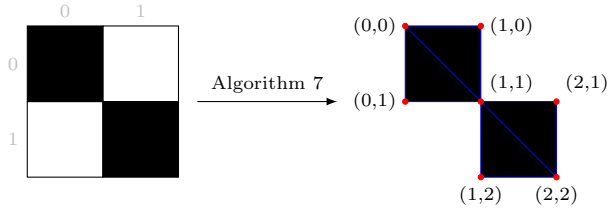

Fig. 2. On the left, a digital image; on the right, its simplicial complex representation

Once we have the simplicial complex associated with the digital image, we can compute the homology groups of the image from the simplicial complex. As we said previously, several simplicial complexes can be associated with a digital image, but all of them are homomorphic (see [1]); then, we can define the homology groups of a 2D-image as follows:

Definition 9 Given a 2D-image $\mathcal{D}$, the *n-homology group* of $\mathcal{D}$, $H_n(\mathcal{D})$ is the n-homology group of the simplicial complex $\mathcal{K}_{2D}(\mathcal{D})$:

$$H_n(\mathcal{D}) = H_n(\mathcal{K}_{2D}(\mathcal{D})).$$

Subsequently, we can interpret properties about the digital image from its homology groups. 2D-images are embedded in $\mathbb{R}^2$ then its homology groups vanish for dimensions greater than 2 and they are torsion-free from dimensions 0 to dimension 1; that is, their homology groups are either null or a direct sum of $\mathbb{Z}$ components from dimension 0 to dimension 1. The number of $\mathbb{Z}$ components of the homology groups of dimension 0 and 1 measures respectively the number of connected components and the number of holes of the image.

The method presented here for 2D-images can be generalized to nD-images with $n \geq 2$. An nD-image can be represented by a finite n-dimensional array of 1's and 0's in which each 1 represents an n-xel in $\mathcal{D}$ and each 0 represents an n-xel that is not in $\mathcal{D}$.

Let $\mathcal{D}$ be an nD-image, from the coordinates of each n-xel in $\mathcal{D}$ (its position in the n-dimensional array associated with $\mathcal{D}$), we can obtain a triangulation by means of n-simplexes, see [12], which are facets of a simplicial complex associated with $\mathcal{D}$. If we repeat the process for the coordinates of all the n-xels in $\mathcal{D}$, we obtain the facets of a simplicial complex associated with $\mathcal{D}$. Then, applying Algorithm 6, we can obtain the simplicial complex associated with $\mathcal{D}$. Therefore, the two following algorithms can be defined.

Algorithm 10
Input: the coordinates of an n-xel.
Output: a triangulation of the n-xel by means of n-simplexes.

Algorithm 11
Input: an nD-image $\mathcal{D}$ represented by means of a n-dimensional array of 1's and 0's.
Output: the facets of $\mathcal{K}_{nD}(\mathcal{D})$, a simplicial complex associated with $\mathcal{D}$.

4 Formalization in the ACL2 Theorem Prover

As we just said, we want to formalize in ACL2 the correctness of Algorithm 7; namely, our implementation of that algorithm by means of a Common Lisp function called `genera-facets-image-2d`.

From now on, we define the necessary functions to establish the correctness of our program. First of all, we need some auxiliary functions which define the necessary concepts to prove our theorems. These definitions are based on both Algorithm 7 and the notions for digital images. Namely, we need to define the notion of 2D-image.

As we said in Section 3, a 2D-image $\mathcal{D}$ is represented by means of a finite 2-dimensional array (that is a list of lists) of 1's and 0's where each 1 represents a pixel in $\mathcal{D}$ and each 0 represents a pixel that is not in $\mathcal{D}$. The `2d-imagep` function

is a function that checks if its argument is a list of lists of 1's and 0's. This function uses the `list-0-1-p` function that checks if its argument is a list of 1's and 0's.

```
(defun list-0-1-p (list)
  (if (endp list)
      (equal list nil)
    (if (endp (cdr list))
        (and (equal (cdr list) nil)
             (or (equal (car list) 0) (equal (car list) 1)))
      (and (or (equal (car list) 0) (equal (car list) 1))
           (list-0-1-p (cdr list))))))
```

```
(defun 2d-imagep (list)
  (if (endp list)
      (equal list nil)
    (and (list-0-1-p (car list)) (2d-imagep (cdr list)))))
```

Subsequently, we define the `genera-facets-image-2d` function and all its auxiliary functions in ACL2. Let us show in detail these definitions.

First of all, we define the `list-up-i-j` and `list-down-i-j` functions which are used to generate from a pair of natural numbers (i, j) the simplexes $((i, j), (i + 1, j), (i + 1, j + 1))$ and $((i, j), (i, j + 1), (i + 1, j + 1))$ respectively.

```
(defun list-up-i-j (i j)
  (list (list i j) (list (1+ i) j) (list (1+ i) (1+ j))))
```

```
(defun list-down-i-j (i j)
  (list (list i j) (list i (1+ j)) (list (1+ i) (1+ j))))
```

From the above two functions, we can define a function, called `genera-facets-i-j` which from the pair (i, j) generates the pair of simplexes $(((i, j), (i + 1, j), (i + 1, j + 1)), ((i, j), (i, j + 1), (i + 1, j + 1)))$

```
(defun genera-facets-i-j (i j)
  (list (list-up-i-j i j) (list-down-i-j i j)))
```

Now, we can define the `genera-facets-image-2d` function which generates the simplexes of a list of lists of 0's and 1's `lol`.

```
(defun genera-facets-image-2d (lol)
  (genera-facets-image-aux lol 0))
```

The above function calls the more general function `genera-facets-image-aux` which takes two arguments: a list of lists of 0's and 1's `lol` and a natural number j. The function `genera-facets-image-aux` must be understood as the procedure which generates the simplexes of the list of lists of 0's and 1's `lol` which is the sublist located from position j of another list of lists of 0's and 1's, let us called it `lol-main`.

```
(defun genera-facets-image-aux (lol j)
  (if (endp lol)
      nil
    (append (genera-facets-list (car lol) 0 j)
            (genera-facets-image-aux (cdr lol) (1+ j)))))
```

For each one of the lists of 0's and 1's of `lol` and the position of that list in `lol-main`, the above function invokes the function `genera-facets-list`. The function `genera-facets-list` must be understood as the procedure which generates the simplexes of the list of 0's and 1's `list` which is the sublist located from position i of the list of position j of `lol-main`.

```
(defun genera-facets-list (list i j)
  (if (endp list)
      nil
    (if (equal (car list) 1)
        (append (genera-facets-i-j i j) (genera-facets-list (cdr list) (1+ i) j))
      (genera-facets-list (cdr list) (1+ i) j))))
```

Once we have defined our programs in ACL2 we can prove theorems about them. To be more concrete, we have proved both the correctness a the completeness of our program `genera-facets-image-2d`.

First of all we state the ACL2 theorem which ensures the completeness of `genera-facets-image-2d`.

ACL2 Theorem 12 Let `image` be a 2D-image represented by means of a 2 dimensional array, then, $\forall i, j \in \mathbb{N}$ such that the value of the image in position (i, j) of the array is 1, then, the simplexes $((i, j), (i+1, j), (i+1, j+1))$ and $((i, j), (i, j+1), (i+1, j+1))$ are in the list generated by the `genera-facets-image-2d` function taking as input `image`.

To state this theorem in ACL2, we need the ACL2 functions: `(natp n)`, which is a test function returning `t` if `n` is a natural number and `nil` otherwise; `(nth i ls)`, which returns the value of position `i` (a natural number) of the list `ls`; and, `(member-equal x ls)`, which returns `t` if `x` is equal to some of the elements of `ls` (a list).

```
(defthm genera-facets-image-2d-completeness
  (implies (and (2d-imagep image)
                (natp i)
                (natp j)
                (equal (nth i (nth j image)) 1))
           (and (member-equal (list-up-i-j i j) (genera-facets-image-2d image))
                (member-equal (list-down-i-j i j) (genera-facets-image-2d image)))))
```

Once we have proved the completeness of our program, we must prove its correctness. This task is handled by means of the following lemmas.

ACL2 Theorem 13 Let `image` be a 2D-image represented by means of a 2 dimensional array and `simplex` be an element of the output generated by `genera-facets-image-2d` taking as input `image`. Then if `simplex` is of the form $((i, j), (i+1, j), (i+1, j+1))$

with i and j natural numbers, then the element $((i,j),(i,j+1),(i+1,j+1))$ is also in the output generated by `genera-facets-image-2d` taking as input `image`.

To state this theorem in ACL2 we need some auxiliary functions. Namely, `member-list-up`, which returns `t` if its input is a list of the form $((i,j),(i+1,j),(i+1,j+1))$ and `nil` otherwise; and `list-down`, which from a list of the form $((i,j),(i+1,j),(i+1,j+1))$ returns the list $((i,j),(i,j+1),(i+1,j+1))$.

```
(defthm genera-facets-image-correctness-1
  (implies (and (2d-imagep image)
                (member-equal simplex (genera-facets-image-2d image))
                (member-list-up simplex))
           (member-equal (list-down simplex) (genera-facets-image-2d image))))
```

ACL2 Theorem 14 Let `image` be a 2D-image represented by means of a 2 dimensional array and `simplex` be an element of the output generated by `genera-facets-image-2d` taking as input `image`. Then if `simplex` is of the form $((i,j),(i,j+1),(i+1,j+1))$ with i and j natural numbers, then the element $((i,j),(i+1,j),(i+1,j+1))$ is also in the output generated by `genera-facets-image-2d` taking as input `image`.

To state this theorem in ACL2 we need some auxiliary functions. Namely, `member-list-down`, which returns `t` if its input is a list of the form $((i,j),(i,j+1),(i+1,j+1))$ and `nil` otherwise; and `list-up`, which from a list of the form $((i,j),(i,j+1),(i+1,j+1))$ returns the list $((i,j),(i+1,j),(i+1,j+1))$.

```
(defthm genera-facets-image-correctness-2
  (implies (and (2d-imagep image)
                (member-equal simplex (genera-facets-image-2d image))
                (member-list-down simplex))
           (member-equal (list-up simplex) (genera-facets-image-2d image))))
```

ACL2 Theorem 15 Let `image` be a 2D-image represented by means of a 2 dimensional array and `simplex` be an element of the output generated by `genera-facets-image-2d` taking as input `image` of the form $((i,j),(i+1,j),(i+1,j+1))$ or $((i,j),(i,j+1),(i+1,j+1))$ with i and j natural numbers. Then, the element of position (i,j) of `image` is 1.

To state this theorem in ACL2 we use some ACL2 functions which have not been used previously: `caar` which returns the first element of the first element of a list and `cadar` which returns the second element of the first element of a list.

```
(defthm genera-facets-image-correctness-3
  (implies (and (2d-imagep image)
                (member-equal simplex (genera-facets-image-2d image)))
           (equal (nth (caar simplex) (nth (cadar simplex) image)) 1)))
```

Let us present some remarks about the proof of these theorems which state both the completeness and the correctness of the `genera-facets-image-2d` program.

First of all, it is worthwhile noting that the implementation of the `genera-facets-image-2d` function, and its auxiliar ones, follows simple recursive schemas, that are suitable for the induction heuristics of the ACL2 theorem prover.

Let us present now some particularity of the auxiliary lemmas needed in our development of the proof of the main theorems. As we have seen in the definition of `genera-facets-image-aux`, this function invokes the `genera-facets-list` function with arguments `(car list)`, `0` and `j`. Therefore, it is sensible to think that in the proof of our theorems we are going to need some auxiliary lemmas such as:

```
(thm (implies (and (list-0-1 x) (natp j)
                   (member-equal simplex (genera-facets-list x 0 j)))
              (equal (nth (caar simplex) x) 1))
```

that is to say, a lemma which involves a call to `(genera-facets-list x 0 j)`. However, ACL2 has some problems to find a proof of theorems such as the previous one, since it does not find a good inductive schema for reasoning. On the contrary, for ACL2 is much more easier to find a proof of theorems such as:

```
(thm (implies (and (list-0-1 x) (natp i) (natp j)
                   (member-equal simplex (genera-facets-list x i j)))
              (equal (nth (- (caar simplex) i) x) 1))
```

that is to say, lemmas that are general cases of the previous ones.

Taking this question into account in the development of our proofs, the certification of the completeness and correctness theorems does not mean any special trouble.

In this way, we have proved the completeness and correctness of our implementation of Algorithm 7 by means of the program `genera-facets-image-2d`.

In the case of 3D-digital images the development is very similar and has not involve any special hidrance.

5 Formalization in Coq/SSReflect

For the formalization of Algorithm 7 in Coq, we decided to use the libraries already provided in the SSReflect library.

First of all, we define the notion of digital image. A digital image is defined as a sequence of sequences of boolean elements.

```
Definition image : Type := seq (seq bool).
```

Following, the same schema presented in the case of ACL2, we can define a function which implements Algorithm 7.

```
Definition genera_facets_image_2d (image: image) :=
  genera_facets_image_aux image 0.
```

The ACL2 theorems presented in the previous section have been also formalized in Coq/SSReflect.

SSReflect Theorem 16 Let `image` be a 2D-image represented by means of a 2 dimensional array, then, $\forall i, j \in \mathbb{N}$ such that the value of the image in position (i, j) of the array is 1, then, the simplexes $((i, j), (i + 1, j), (i + 1, j + 1))$ and

$((i,j),(i,j{+}1),(i{+}1,j{+}1))$ are in the list generated by the `genera_facets_image_2d` function taking as input `image`.

```
Lemma genera_facets_image_2d_completeness : forall (image: image)
    (i j:nat),
  (nth nil (nth nil im i) j) -> ((list_up_i_j i j) \in (
      genera_facets_image_2d image)) && ((list_down_i_j i j) \in (
      genera_facets_image_2d image)).
```

SSReflect Theorem 17 Let `image` be a 2D-image represented by means of a 2 dimensional array and `simplex` be an element of the output generated by `genera_facets_image_2d` taking as input `image`. Then if `simplex` is of the form $((i,j),(i+1,j),(i+1,j+1))$ with i and j natural numbers, then the element $((i,j),(i,j{+}1),(i{+}1,j{+}1))$ is also in the output generated by `genera_facets_image_2d` taking as input `image`.

```
Lemma genera_facets_image_correctness_1 : forall (i j:nat) (image:
     image): ((list_up_i_j i j)\in (genera_facets_image_2d image))
     -> ((list_down_i_j i j)\in (genera_facets_image_2d image)).
```

SSReflect Theorem 18 Let `image` be a 2D-image represented by means of a 2 dimensional array and `simplex` be an element of the output generated by `genera-facets-image-2d` taking as input `image`. Then if `simplex` is of the form $((i,j),(i,j+1),(i+1,j+1))$ with i and j natural numbers, then the element $((i,j),(i{+}1,j),(i{+}1,j{+}1))$ is also in the output generated by `genera-facets-image-2d` taking as input `image`.

```
Lemma genera_facets_image_correctness_2 : forall (i j:nat) (image:
     image): ((list_down_i_j i j)\in (genera_facets_image_2d image
    )) -> ((list_up_i_j i j)\in (genera_facets_image_2d image)).
```

SSReflect Theorem 19 Let `image` be a 2D-image represented by means of a 2 dimensional array and `simplex` be an element of the output generated by `genera-facets-image-2d` taking as input `image` of the form $((i,j),(i+1,j),(i+1,j+1))$ or $((i,j),(i,j+1),(i+1,j+1))$ with i and j natural numbers. Then, the element of position (i,j) of `image` is 1.

```
Lemma genera_facets_image_correctness_3 : forall (i j:nat) (image:
     image): ((list_up_i_j i j) \in (genera_facets_image_2d image)
    ) -> (nth nil (nth nil s i) j).
```

In this way, we have proved the correctness of Algorithm 7 in Coq.

6 Conclusions and further work

In this report, we have presented a methodology to study digital images by means of simplicial complexes. Moreover, we have formalized in both the ACL2 Theorem Prover and the proof assistant Coq of the correctness of our algorithms.

As further work we can undertake the task of formalizing reduction algorithms for digital images which keep topological properties: on the one hand, we can verify algorithms which reduce the original image; on the other hand, we can formalize the reduction of simplicial complexes applying techniques such as Morse theory [13].

Another topic is related to the study of the feasibility of applying our methods to the study of real medical images.

References

1. R. Ayala, E. Domínguez, A.R. Francés, and A. Quintero. Homotopy in digital spaces. *Discrete Applied Mathematics*, 125:3–24, 2003.
2. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development, Coq'Art: the Calculus of Inductive Constructions*. Springer-Verlag, 2004.
3. Y. Bertot, L. Rideau, and ForMath La Rioja node. Report on a ssreflect week. Technical report, 2010. `http://wiki.portal.chalmers.se/cse/uploads/ForMath/SSReflectWeekLaRioja`.
4. Coq development team. The Coq Proof Assistant Reference Manual, version 8.3. Technical report, 2010.
5. G. Gonthier and A. Mahboubi. A Small Scale Reflection Extension for the Coq system. Technical report, Microsoft Research INRIA, 2009. `http://hal.inria.fr/inria-00258384`.
6. R. González-Díaz, B. Medrano, P. Real, and J. Sánchez-Peláez. Algebraic Topological Analysis of Time-Sequence of Digital Images. In *Proceedings 8th International Conference on Computer Algebra in Scientific Computing (CASC'2005)*, volume 3718, 2005.
7. R. González-Díaz and P. Real. On the Cohomology of 3D Digital Images. *Discrete Applied Math*, 147(2-3):245–263, 2005.
8. L. J. Hernández. Orografía homológica cúbica para la minería de datos. Technical report, University of La Rioja. `http://www.unirioja.es/cu/luhernan/datamining`.
9. M. Kaufmann and J. S. Moore. ACL2 version 3.4, 2009. `http://www.cs.utexas.edu/users/moore/acl2/`.
10. D. Mackenzie. Topologists and Roboticists Explore and Inchoate World. *Science*, 8:756, 2003.
11. J. P. May. *Simplicial objects in Algebraic Topology*, volume 11 of *Van Nostrand Mathematical Studies*. 1967.
12. D. Orden and F. Santos. Asymptotically efficient triangulations of the d-cube. *Discrete and Computational Geometry*, 30(4):509–528, 2003.
13. A. Romero and F. Sergeraert. Discrete vector fields and fundamental algebraic topology. Technical report, Institut Fourier, 2010. `http://www-fourier.ujf-grenoble.fr/~sergerar/Papers/Vector-Fields.pdf`.

14. F. Ségonne, E. Grimson, and B. Fischl. Topological Correction of Subcortical Segmentation. In *Proceedings 6th International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI'2003)*, volume 2879 of *Lecture Notes in Computer Science*, pages 695–702, 2003.
15. J. Wood. Spinor groups and algebraic coding theory. *Journal of Combinatorial Theory*, 50:277–313, 1989.