Coding cells of digital spaces: a framework to write generic digital topology algorithms

Jacques-Olivier Lachaud [a]

[a] *LaBRI, Univ. Bordeaux 1, 351 cours de la Libération, 33405 Talence, France*

Abstract

This paper proposes a concise coding of the cells of n-dimensional finite regular grids. It induces a simple, generic and efficient framework for implementing classical digital topology data structures and algorithms. Discrete subsets of multidimensional images (e.g. regions, digital surfaces, cubical cell complexes) have then a common and compact representation. Moreover, algorithms have a straightforward and efficient implementation, which is independent from the dimension or sizes of digital images. We illustrate that point with generic hypersurface boundary extraction algorithms by scanning or tracking. This framework has been implemented and basic operations as well as the presented applications have been benchmarked.

1 Introduction

Many applications in the image analysis field need to represent and manipulate discrete subsets of digital spaces. As the image data become larger, the data structures required to represent these sets should be as compact as possible. Moreover, algorithms designed on these structures should be not only efficient theoretically, but also efficient in practice. Algorithms defined formally should have a straightforward implementation. Last but not least, 3D and 4D image datasets are now more and more common. It becomes necessary to have a unified framework for programming applications dealing with n-dimensional data. By this way, algorithms are both generically defined and implemented.

There exist several approaches to define the topology of multidimensional regular digital spaces: (1) adjacency graphs as pioneered by Rosenfeld, (2) oriented graphs as proposed by Herman [5], (3) cellular complexes as proposed by Kovalevsky [11], or equivalently Khalimsky's spaces [7] and interpixel representations. This paper deals mostly with the third approach, although our framework can express either approaches (the two first approaches manipulate a restricted set of the elements defined in the spaces of the third approach).

The cellular decomposition of the Euclidean n-dimensional space $\mathbb{R}^n$ into a regular grid forms a cellular complex $\mathbb{C}^n$. This structure has been introduced in digital topology by Kovalevsky [11] for 2D and 3D applications. It has been shown that the topology induced on $\mathbb{C}^n$ is equivalent to a digital topology $\mathbb{K}^n$, sometimes called *Khalimsky* topology [7,8]. Many authors have explored the theoretical properties of the space $\mathbb{C}^n$ (or equivalently $\mathbb{K}^n$) [7,8,11] applications [2,3,9,10,14]. These works show that the definition of consistent high level data structure over images relies on a low-level representation which is the regular cellular decomposition of the image support. It is thus critical to represent efficiently arbitrary cells of $\mathbb{C}^n$, small subsets of $\mathbb{C}^n$ and specific subsets of $\mathbb{C}^n$ (e.g. complexes, digital surfaces). However, the litterature does not reflect this observation. Indeed, spels are often coded with an array of coordinates, surfels as pairs of adjacent spels or a spel with a direction, other cells are generally implicitly represented. Consequently, storing elements or subsets of $\mathbb{C}^n$ is cumbersome; algorithms are frequently rewritten at the implementation stage to avoid any reference to non elementary kinds of cells.

In this paper, we choose another approach, which is first to show how to represent an arbitrary cell of $\mathbb{C}^n$ with a binary cell code and secondly to design data structures over this representation. Because of the regularity of $\mathbb{C}^n$, each cell code holds all the information on the cell: the cell topology (dimension, open or closed along a coordinate, adjacent and incident cells) and geometry (coordinates in $\mathbb{Z}^n$, centroid, normal and tangent vectors) can be computed from the code without any other information. The proposed framework is suited both to formal representation and proofs and to straightforward implementation in a programming language. The paper is organized as follows: (i) coding of cells and implementation of low-level digital topology definitions, (ii) definition of data structures for subsets of $\mathbb{C}^n$ (e.g., digital surfaces, complexes), (iii) application to digital boundaries extraction in multidimensional images. We emphasize that all operations and algorithms have the same definitions and implementation whatever the dimension of the space. All experiments and benchmarks presented in this paper were made on a PC with a Celeron 500Mhz processor, 128Mb of memory, 128Kb of cache (which is a basic workstation). The proposed framework was implemented in C++. Due to limited space, the reader is referred to [12] for more details.

2 Coding cells of digital spaces $\mathbb{C}^n$

2.1 Cellular decomposition $\mathbb{C}^n$; binary coding of unoriented cells

We denote by $\mathbb{C}^n$ the set of parts of the n-dimensional Euclidean space $\mathbb{R}^n$ such that $c \in \mathbb{C}^n$ is equivalent to $c = I_1 \times \ldots \times I_n$ where I_i is a subset of $\mathbb{R}$ of

the form $\{z_i\}$ or $]z_i, z_i+1[$ with $z_i \in \mathbb{Z}$. The complex $\mathbb{C}^n$ is a partition of $\mathbb{R}^n$. We call *k-cell* an element $c \in \mathbb{C}^n$ such that c has k I_i of the form $]z_i, z_i+1[$ (and therefore $(n-k)$ I_i of the form $\{z_i\}$). The *dimension* of c is k. The *closure* $\mathrm{Cl}(c)$ of a cell c is the set of cells c' of $\mathbb{C}^n$ which have the following form: (i) on coordinate where c is a point $\{z_i\}$, the cell c' must also be the same point, (ii) on coordinate where c is an open segment $]z_i, z_i+1[$, c' can be either the same open segment or the point $\{z_i\}$ or the point $\{z_i+1\}$. The *open star* $\mathrm{Op}(c)$ of a cell c is the set of cells c' of $\mathbb{C}^n$ such that $c' \in \mathrm{Op}(c) \Leftrightarrow c \in \mathrm{Cl}(c')$. The *bounding relation* $<$ between two cells c and c' is then defined as $c < c'$ iff $c \in \mathrm{Cl}(c') \setminus c'$. With these definitions, the set $\mathbb{C}^n$ equipped with the dimension mapping and the bounding relation is a cellular complex. A *cubical cell complex* K is then defined as any set of cells in a finite image. The *dimension* of K is the maximum of the dimensions of its cells. Open stars and closure of cells in a complex K are defined naturally.

It is clear that the elements of $\mathbb{C}^n$ represent low-level elements of n-dimensional digital images: the *spels* (pixels in 2D and voxels in 3D) are the n-cells, the *(unoriented) surfels* (a pair of adjacent spels) are the $n-1$-cells, the vertices of the spels and of the surfels, or *pointels*, are the 0-cells. An *object* is then a set of n-cells, a *digital surface* is a set of $n-1$-cells (oriented or not, see Section 2.2), a curve is a set of connected 1-cells. Therefore, all classical subsets of digital spaces have a natural definition as specific subsets of $\mathbb{C}^n$. From now on, we will assume that we are working in a finite n-dimensional image forming a parallelepiped in $\mathbb{Z}^n$. We denote by M^i the inclusive upper bound for the i-th coordinate of any spel. All coordinates have 0 as lower bound.

As shown by Kong et al. [8], the topology of $\mathbb{C}^n$ is equivalent to the topology of the Khalimsky digital space $\mathbb{K}^n$, which is the cartesian product of n connected ordered topological spaces (COTS). A COTS can be seen as a set of ordered discrete points, like $\mathbb{Z}$, whose topology alternates closed points and open points. If we define even points of $\mathbb{Z}$ as closed and odd points of $\mathbb{Z}$ as open, each point of $\mathbb{K}^n$ is then identified by its n integer coordinates, whose parities define its topological properties.

Consequently, any k-cell c of $\mathbb{C}^n$ has exactly one corresponding point in $\mathbb{K}^n$ with coordinates $(x_K^0, \ldots, x_K^{n-1})$. We propose to code c as one binary word $\mathtt{code}(c) = \boxed{\alpha}\boxed{x^{n-1}}\boxed{\ldots}\boxed{x^i}\boxed{\ldots}\boxed{x^0}$, called the *unsigned code* of c, as follows:

- The i-th coordinate x_K^i is coded by its binary decomposition after a right-shift ($x^i = x_K^i$ div 2). We say that x^i is the *i-th digital coordinate* of c.
- All coordinates are packed as one binary word (from x^{n-1} to x^0). Every coordinate is allocated a fixed number of bits N_i given by $N_i = \log_2(M^i)+1$.
- The parity of all coordinates are also packed as an n-bits word α with $\alpha = \sum_i (x_K^i \bmod 2)2^i$. α is called the *topology* of c.

Spels have a topology word composed of 1's, whereas pointels have a topology word made of 0's. Surfels have one 0 and $n-1$ 1's in their topology word. The coordinate where a surfel c has a 0 in its topology word is called the coordinate *orthogonal* to the surfel c and is denoted by $\perp(c)$. This coding implies that any cell of finite digital images can be coded as an integer number with fixed size. Any register of a processor may thus store a cell if the image is not too big. [1] We define the adjacency between cells independently of the cell topology.

Definition 1 *Two cells p and q with $\texttt{topo}(p) = \texttt{topo}(q)$ are* l-adjacent *if their respective coordinates p^i and q^i differ by at most 1 and if the infinite norm of the vector $(p^{n-1} - q^{n-1}, \ldots, p^0 - q^0)$ is no more than l.*

The 1-adjacency thus defines the 4-adjacency (resp. 6-adjacency) on pixels in 2D (resp. on voxels in 3D) and the 2-adjacency defines the 8-adjacency (resp. 18-adjacency) on pixels in 2D (resp. on voxels in 3D). We define the incidence relation as below. The proposition that follows shows that all the topological structure of $\mathbb{C}^n$ can be obtained from the incidence relation.

Definition 2 *Let $c = \boxed{\alpha|x^{n-1}|\ldots|x^i|\ldots|x^0}$ be a cell and i any coordinate. Let $\beta = \alpha$ xor 2^i. If the i-th bit of α is set to 1, the cell c has two* low 1-incident cells *along coordinate i coded by $\boxed{\beta|x^{n-1}|\ldots|x^i|\ldots|x^0}$ and $\boxed{\beta|x^{n-1}|\ldots|x^i+1|\ldots|x^0}$. Otherwise, if the i-th bit of α is set to 0, the cell c has two* up 1-incident cells *along coordinate i coded by $\boxed{\beta|x^{n-1}|\ldots|x^i-1|\ldots|x^0}$ and $\boxed{\beta|x^{n-1}|\ldots|x^i|\ldots|x^0}$. A cell p is* low incident *(resp.* up incident*) to a cell q if there is a sequence of cells $c_0 = p, c_1, \ldots, c_k = q$ such that $\forall j$, c_j is low 1-incident (resp. up 1-incident) to c_{j+1}.*

Proposition 3 *The set of cells low incident to a cell c is equal to $\mathrm{Cl}(c) \setminus c$. The set of cells up incident to c is equal to $\mathrm{Op}(c) \setminus c$.*

Figure 1 summarizes the number of elementary operations necessary to execute basic cell operations. Their implementation is fully generic. We have benchmarked these operations and the results show that cell codes compete with statically defined structures (e.g. fixed size arrays) and are much faster than dynamically allocated structures, classically used for generic programming.

2.2 Oriented cells, boundary operators, bels, boundary of an object

In some applications, it is convenient to orient the cells (as positive or negative). For instance, digital surfaces as proposed by Herman and Udupa are

[1] 32 bits are sufficient to code every cell of a $1024 \times 1024 \times 512$ 3D image, which is more than enough for current biomedical applications.

nb ops required	code	topo, coord	==	set coord	adj.	is l-adj.?	inc.	is l-inc.?
bits ops	0	1	0	2	0	$\leq 2n$	1	≤ 3
shifts	n	1	0	1	0	0	0	≤ 6
integer ops	n	0	1	0	1	$\leq 2n$	≤ 1	$\leq l+4$
lut access	n	1	0	2	1	$\leq n$	≤ 2	$\leq l+2$
cond. tests	0	0	0	0	1	$\leq 2n$	1	$\leq 3l+1$

Fig. 1. Number of elementary operations needed to perform the following tasks: (i) coding a vector of n Khalimsky coordinates as a cell, (ii) getting the topology or one coordinate of a cell, (iii) comparing if two cells are identical, (iv) setting the coordinate of a cell, (v) computing a 1-adjacent cell, (vi) checking if two cells are l-adjacent, (vii) computing a 1-incident cell, (viii) checking if two cells are l-incident.

composed of oriented pairs of voxels: one voxel is in the interior of the surface, the other in the exterior. Orienting a surfel means in this case to define where are the interior and exterior voxels 1-up-incident to the surfel. Digital surface tracking algorithms rely on this orientation for a consistent output. Classical combinatorial topology associates an orientation to each cell of a cellular complex. Oriented cells are then useful to implement boundary operators over complexes and to compute topological invariants.

We therefore define the *signed code* of a cell c with orientation bit s (0 is positive, 1 is negative) by adding the bit s between the topology α of c and its digital coordinates x^i as follows: $\boxed{\alpha}\boxed{s}\boxed{x^{n-1}}\boxed{\ldots}\boxed{x^i}\boxed{\ldots}\boxed{x^0}$. The *opposite cell* $-c$ of c is the same cell as c but with opposite sign. Boundary operators, which can be seen informally as an oriented version of incidence, are essential in combinatorial topology : for instance, they define the topology of polyhedral complexes. We have now to "orient" the incidence relation.

Definition 4 *Let* $c = \boxed{i_k \ldots i_j \ldots i_0}\boxed{s}\boxed{x^{n-1}}\boxed{\ldots}\boxed{x^{i_j}}\boxed{\ldots}\boxed{x^0}$ *be any cell with topology bits set to 1 on the coordinates* $i_k, \ldots, i_j, \ldots, i_0$, $n-1 \geq i_k > \ldots > i_j > \cdots > i_0 \geq 0$ *and the others bits set to 0. The symbol* $\hat{i}_j$ *means that the bit* i_j *is set to 0. Let* $\tau = (-1)^{(k-j)}$. *The set* $\Delta_{i_j} c$ *composed of the two oppositely signed cells* $\tau\boxed{i_k \ldots \hat{i}_j \ldots i_0}\boxed{s}\boxed{x^{n-1}}\boxed{\ldots}\boxed{x^{i_j}}\boxed{\ldots}\boxed{x^0}$ *and* $-\tau\boxed{i_k \ldots \hat{i}_j \ldots i_0}\boxed{s}\boxed{x^{n-1}}\boxed{\ldots}\boxed{x^{i_j}+1}\boxed{\ldots}\boxed{x^0}$, *is called the* lower boundary of the cell c along coordinate i_j. *The* lower boundary Δc *of* c *is then the set of cells* $\cup_{l=0,\ldots,k} \Delta_{i_l} c$.

The lower boundary of c thus corresponds to the set of cells 1-low-incident to c with specific orientations. The *upper boundary* ∇ of a cell is defined symmetrically (the upper boundary is taken on topology bits set to 0). It can be shown that this definition of boundary operators induces that any cubical cell complex is a polyhedral complex. Although this is outside the scope of this paper,

boundary operators along with *chains* are used to define the (co)homology groups of complexes, which are well known topological invariants. This topological tool is readily applicable to cubical cell complexes.

In the remainder of the paper, the set O is an object of the image I with an empty intersection with the border of I. Assume that all spels of O are oriented positively. We merge the sets Δp with $p \in O$ with the rule that two identical cells except for their orientation cancel each other. The resulting set of oriented surfels is called the *boundary of* O, denoted by ∂O. It is a digital surface, whose elements are called *bels* of O. The following result states that the boundary of O is indeed the digital surface separating the spels of O from the spels of the complement of O.

Proposition 5 *Let c be a bel of ∂O. Then ∇c contains two spels: one positively oriented and belonging to O, the other negatively oriented and not belonging to O. Any path of 1-adjacent spels from an element of O to an element not in O crosses ∂O.*

One way to compute the digital surface bordering a set of spels O is by applying the lower boundary operator on each element of O and removing oppositely oriented identical cells. The time complexity of this algorithm is thus linear with the number of spels of O.

2.3 Followers of surfel, bel adjacency, digital surface tracking

The bel adjacency defines the connectedness relations between bels bounding an object. It has two nice consequences: (i) the boundary of an object can be extracted by tracking the bels throughout their bel adjacencies [1,4]; (ii) sets of surfels can be considered as classical Euclidean surfaces, where one can move on the surface in different orthogonal directions (2 in 3D). The second reason is essential for defining the geometry of digital surfaces [12]. We present here a definition of bel adjacencies that is essentially equivalent to the definition of [6], but easier to implement in our framework. We start by defining which surfels are potentially adjacent to a given bel with the notion of follower. We then define two kinds of bel adjacency for each pair of coordinates.

Definition 6 *We say that an r-cell q is a* direct follower *of an r-cell p, $p \neq \pm q$, if Δp and Δq have a common $r-1$-cell, called the* direct link *from p to q, such that this cell is positively oriented in Δp and negatively oriented in Δq. The cell p is then an* indirect follower *of q.*

It is easy to check that any surfel has 3 direct followers and 3 indirect followers along all coordinates except the one orthogonal to the surfel. We order the followers consistently for digital surface tracking (see Figure 2).

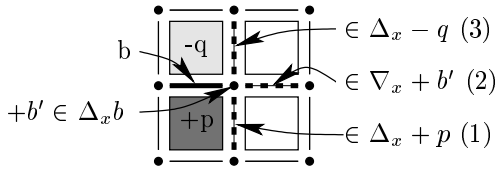

Fig. 2. Direct followers of a surfel b along coordinate x.

Definition 7 *Let b be an oriented $n-1$-cell, such that $\nabla b = \{+p, -q\}$. Let j be a coordinate with $j \neq_\perp (p)$. The three direct followers of p along j are ordered as follows: (1) the first direct follower belongs to $\Delta_j + p$, (2) the second direct follower belongs to $\nabla_j + b'$ with $+b'$ direct link in $\Delta_j b$, (3) the third direct follower belongs to $\Delta_j - q$.*

Intuitively, when tracking a digital surface, you have 3 different possibilities for a move along a given coordinate. This is true for arbitrary dimension. The following definition shows which one to choose at each step . It is in agreement with the definitions of bel adjacencies proposed by Udupa [15].

Definition 8 *Let b be a bel of ∂O, such that $\nabla b = \{+p, -q\}$ (thus $p \in O$ and $q \notin O$). For any coordinate $j \neq_\perp (b)$, the bel b has one* interior direct adjacent bel *(resp.* exterior direct adjacent bel*) which is the first (resp. last) of the three ordered direct followers of b along coordinate j that is a bel of O. The bel adjacency is the symmetric closure of the direct bel adjacency.*

In 3D, the interior (resp. exterior) bel adjacency along all coordinates induces the classical (6,18) bel-adjacency (resp. (18,6) bel-adjacency). Interior and exterior bel adjacencies can be mixed for different coordinate pairs. This might be useful in an application where the image data are not isotropic (e.g., some CT scan images, confocal microscopy). Computing the bel adjacent to a given one is very fast since it required [12]: ≤ 11 binary or integer operations, ≤ 3 shifts, ≤ 14 lut accesses, ≤ 9 conditional tests, and 1 or 2 "is in set" operations. The next section will show that the "is in set" operation can be done in four elementary operations.

The following theorem, which comes from the fact that cubical cell complexes are polyhedral complexes, is interesting to speed up digital surface tracking algorithm: as its corollary, tracking only direct adjacent bels is sufficient to extract the whole digital surface component that contains the seed bel. It is more complex to show that bel components correspond to interior and exterior components of spels (see [13,15] where this is proven for some bel adjacency relations).

Theorem 9 *The transitive closure of the direct bel adjacency from a bel b of ∂O coincides with the transitive closure of the bel adjacency from b.*

Table 1
This table shows some properties of classical set data structures. The symbol + indicates that it is only amortized time complexity.

set structure	STL class	is in set ?	other set ops	memory (bytes)
dynamic array	`vector`	$O(m)$	$O(m)+$	$\approx 4m$
linked list	`list`	$O(m)$	$O(m)$	$\approx 24m$
RB-tree	`set`	$O(\log m)+$	$O(\log m)+$	$\approx 32m$
hashtable	`hash_set`	$O(1)+$	$O(1)+$	$\approx 4m' + 20m$

3 Data structures built over cells

Since any kind of cell is coded as an integer number, data structures coding sets of cells are easily derived from standard data types. Table 1 displays the traditional set data structures, their implementation in C++ as template classes, the time complexities of some operations, and the memory cost. [2]

These data structures are adapted to sets of cells of reasonnable size. Very small sets should be defined as `vector`s. `list`s may be used to represent medium size contours. Other medium size sets should be represented with `set`s or `hash_set`s. If the `hash_set` seems rather efficient for most operations (at least from an asymptotic point of view), it is memory costly: 28 Mbytes are necessary to represent a digital surface with $1,000,000$ bels (and $m' = 2m$). Moreover the memory is very fragmented and the cache is thus not efficient. As it is shown later on digital hypersurface tracking algorithms, amortized constant time does not mean very fast.

We present another data structure to represent a set of cells, which exploits the properties of the cell coding. The size of the data structure is dependent only on the size of the image. The time complexity of all operations is then independent from the number of cells represented. This data structure, called the `CharSet`, is a characteristic function that assigns one bit to each cell of the space. Since we will often manipulate sets of cells that contains specific kinds of cells (e.g.. a digital surface is made of surfels), we present two ways to define this structure.

Definition 10 *A* `MinCharSet` *is an array* `tbl` *of s bits, where s is one plus the difference between the highest possible cell code* `MAX` *and the smallest possible cell code* `min`*. Selecting the bit characteristic of the presence of a given cell c is done with* `tbl[(c-min)>>5]&(1<<(c&0x1f))` *for 32-bits words.*

Definition 11 *A* `LUTCharSet` *is an array* `tbl` *of s bits and a look-up table* `lut`*, where s and* `lut` *are dependent on the set of cells (see Table 2).*

[2] A cell is stored in a 32-bits word. We suppose that 12 bytes are necessary to store information about one dynamically allocated memory area (e.g. holds for Linux)

Table 2
This table defines the way `LUTCharSet`s store various specific sets of cells.

set of cells	topologies α	`lut`(α)	size s (bits)	256^3 image size (Mb)
set of spels	$1\dots1$	0 | $0\dots0$ ($\sum N_i$ bits)	$2^{\sum N_i}$	2
unoriented digital surface	$011\dots1$ $101\dots1$ … $1\dots110$	0 | $0\dots0$ 1 | $0\dots0$ … $n-1$ | $0\dots0$ ($\sum N_i$ bits)	$n2^{\sum N_i}$	6
set of oriented r-cells	$0\dots0011\dots1$ $0\dots0101\dots1$ … $1\dots1100\dots0$	0 | $0\dots0$ 1 | $0\dots0$ … $\binom{n}{r}-1$ | $0\dots0$ ($1+\sum N_i$ bits)	$2\binom{n}{r}2^{\sum N_i}$	12 ($r=1,2$)

Selecting the bit characteristic of the presence of a given cell c is done with `tbl[(lut[topo(c)]+sign_coords(c))>>5]&(1<<(c&0x1f))` *for 32-bits words.*

The `LUTCharSet` is more compact than the `MINCharSet` for some sets of cells (and the higher the dimension the more it is) but the access to the characteristic bit of a cell is a bit slower It is now clear why the bit defining the sign of an oriented cell is inserted between the topology and the coordinates of the cell: with this coding, both `CharSet`s use exactly twice more memory for sets of signed cells compared with sets of unsigned cells.

Knowing if a cell belongs to a `CharSet` or any other atomic set operation (add/remove an element) are $O(1)$ operations. All global set operations (like union, intersection, difference, complement) between `CharSet`s are implemented as standard bit operations between arrays of binary words. Their time complexities are linear in the size of the array. Moreover, the implementation of set operations for any set of cells (arbitrary dimension, set of spels, oriented digital surface, set of r-cells, etc) is done only once as bit operations between arrays of binary words. To give an idea of the efficiency of this representation, inverting a set of spels defined in a 512^3 image takes 0.40s ($134,217,728$ spels, 3ns per spel), difference between two sets of spels in the same image takes 0.80s. Furthermore, an unoriented digital surface in a 256^3 image can hold up to $50,331,648$ surfels for a 8Mb memory cost (or 6Mb for `LUTCharSet`). To conclude this section, unsigned sets are twice less costly to store. They should be used when possible. For instance, digital surfaces that are boundaries of a set of spels are always orientable surface. Digital surface tracking can thus be done with unoriented digital surfaces.

```
// Track (B) algorithm.
// ∂O must be closed.
CharSet
Space::track( CharSet O, Cell b,
              BelAdj A )
{ CharSet S = emptySurfelSet();
  queue<Cell> L; // queue of bels
  L.push( b ); // starting bel
  while ( ! L.empty() ) {
    Cell p = L.pop(); // current bel
    // On all coord where p open
    for ( int j = 0; j < dim(); ++j )
      if (j != orthDir(p)) {
        // Track direct followers
        Cell q = A.directAdj(O,p,j);
        if (! S.isInSet(q)) {
          S.add( q );
          L.push( q );
        }
      }
  }
  return S;
}
```

```
// Track (C) algorithm.
// ∂O must be closed.
CharSet
Space::track( CharSet O, Cell b,
              BelAdj A )
{ CharSet S = emptySurfelSet();
  queue<Cell> L; // queue of bels
  list<Cell> T; // "tail" of bdry
  L.push( b ); // starting bel
  T.multipleInsert( b, dim() - 1 );
  while ( ! L.empty() ) {
    Cell p = L.pop(); // current bel
    for ( int j = 0; j < dim(); ++j )
      if ( j != orthDir( p ) ) {
        Cell q = A.directAdj( O, p, j );
        if ( T.find( q ) ) // already
          T.remove( q ); // extracted
        else {
          S.add( q ); L.push( q );
          T.multipleInsert(q,dim()-2);
  }  } }
  return S; // T is empty at loop end
}
```

Fig. 3. Two digital hypersurface tracking algorithm: the Track (B) algorithm requires an efficient "is in set" operation, the Track (C) algorithm stores the list of cells that will be hit again by the tracking. For the set T in (b), we have tried both `list` and `multiset`. The former was much faster than the later in our experiments.

4 Digital boundary extraction by scanning and tracking

We have implemented several digital hyper-surface extraction algorithms which build the digital surface that is the boundary of a given object O. Scanning algorithms examine every spel neighborhood to detect the presence of a bel. They only require the set O as input. Digital surface tracking algorithms require an initial bel b and a bel adjacency `A` to extract the component of the boundary of O that contains b. As described in Section 2.3, defining the bel adjacency `A` is deciding for each pair of coordinates whether `A` is interior or exterior along this plane. Figure 3 shows how to write generic digital surface tracking algorithms with our framework. The implementation in C++ is very close to the formal specification of the algorithm (see [6]).

In the experiments, the object O was a digital volumic ball. Table 3 lists the running times necessary to extract ∂O for balls of various radii and dimensions. The Scan (A) algorithm scans the whole image to find boundaries. The

Table 3
Running times for several boundary extraction algorithms (see text).

Space size	Rad.	Nb spels	Nb surf.	Scan (A)	Scan (B)	Track (A)	Track (B)	Track (C)
4096^2	2000	12566345	16004	2.07s	2.00s	< 0.01s	< 0.01s	0.01s
128^3	30	113081	16926	0.38s	0.03s	0.01s	0.01s	0.06s
128^3	60	904089	67734	0.39s	0.34s	0.07s	0.05s	0.57s
256^3	120	7236577	271350	3.15s	2.70s	0.36s	0.32s	5.24s
512^3	240	57902533	1085502	25.1s	21.2s	1.88s	1.85s	50.6s
64^4	30	4000425	904648	4.26s	4.00s	1.91s	1.37s	4748s

Scan (B) algorithm scans the parallelepipedic subspace containing the ball. The Track (A) algorithm extracts open or closed boundaries from a starting bel (it follows both direct and indirect bel adjacencies). Track (B) and (C) algorithms extract only closed boundaries from a starting bel (they follow only direct bel adjacencies). All these algorithms are written generically and make no assumption on the dimension of the image. The benchmarks show that scanning algorithms depend on the size of the scanned subspace and that tracking algorithms depend on the number of surfels in ∂O. Running times are excellent since each bel is tracked in $\approx 1.7\mu s$ in 3D (and $\approx 1.5\mu s$ in 4D). Note that Track (B) algorithm is much faster than Track (C) algorithm. This is because `CharSet`s are efficient for the query "is a cell in a given set ?".

5 Conclusion

We have presented a binary coding of every cell of the digital space $\mathbb{C}^n$. This coding contains all the topological and geometric information on the cell. It allows the design and implementation of generic low-level algorithms that deals with subsets of $\mathbb{C}^n$. Compact and efficient data structures can be built with this coding. We illustrated the potential of this framework with a classical digital topology application: boundary extraction. Arbitrary dimensional algorithms are readily implemented in this framework and benchmarks have proved that the resulting code is surprisingly efficient in practice. Other digital topology and geometry applications may be found in [12].

References

[1] E. Artzy, G. Frieder, and G.T. Herman. The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm. *Computer Graphics and Image Processing*, 15:1–24, 1981.

[2] Y. Bertrand, G. Damiand, and C. Fiorio. Topological encoding of 3d segmented images. In G. Borgefors, I. Nyström, and G. Sanniti di Baja, editors, *Proc. of 9th Discrete Geometry for Computer Imagery (DGCI'2000), Uppsala, Sweden*, volume 1953 of *Lecture Notes in Computer Science*, pages 311–324. Springer-Verlag, 2000.

[3] J. P. Braquelaire and J. P. Domenger. Representation of segmented image with discrete geometric maps. *Image and Vision Computing*, 17:715–735, 1999.

[4] D. Gordon and J. K. Udupa. Fast surface tracking in three-dimensional binary images. *Computer Vision, Graphics, and Image Processing*, 45(2):196–241, February 1989.

[5] G. T. Herman. Discrete Multidimensional Jordan Surfaces. *Computer Vision, Graphics, and Image Processing*, 54(6):507–515, November 1992.

[6] G. T. Herman. *Geometry of digital spaces.* Birkhäuser, Boston, 1998.

[7] E. Khalimsky, R. Kopperman, and P. R. Meyer. Computer graphics and connected topologies on finite ordered sets. *Topology and its Applications*, 36:1–17, 1990.

[8] T. Y. Kong, R. D. Kopperman, and P. R. Meyer. A topological approach to digital topology. *Am. Math. Monthly*, 98:901–917, 1991.

[9] V. Kovalevsky. A new means for investigating 3-manifolds. In G. Borgefors, I. Nyström, and G. Sanniti di Baja, editors, *Proc. of 9th Discrete Geometry for Computer Imagery (DGCI'2000), Uppsala, Sweden*, volume 1953 of *Lecture Notes in Computer Science*, pages 57–68. Springer-Verlag, 2000.

[10] V. Kovalevsky. Algorithms and data structures for computer topology. In G. Bertrand, A. Imiya, and R. Klette, editors, *Digital and image geometry*, volume 2243 of *Lecture Notes in Computer Science*, pages 38–58. Springer-Verlag, 2001.

[11] V. A. Kovalevsky. Finite Topology as Applied to Image Analysis. *Computer Vision, Graphics, and Image Processing*, 46(2):141–161, May 1989.

[12] J.-O. Lachaud. Coding cells of multidimensional digital spaces to write generic digital topology and geometry algorithms. Research Report 1283-02, LaBRI, University Bordeaux 1, Talence, France, 2002.

[13] J.-O. Lachaud and A. Montanvert. Continuous analogs of digital boundaries: A topological approach to iso-surfaces. *Graphical Models and Image Processing*, 62:129–164, 2000.

[14] I. Metz. Finding neighbours in d-dimensional binary digital images represented by bintrees. In *Proc. of 4th Discrete Geometry for Computer Imagery (DGCI'94), Grenoble, France*, pages 107–116, 1994.

[15] J.K. Udupa. Multidimensional Digital Boundaries. *CVGIP: Graphical Models and Image Processing*, 56(4):311–323, July 1994.