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

Jacques-Olivier Lachaud

LaBRI - Laboratoire Bordelais de Recherche en Informatique
Bordeaux, France
Outline

- Motivation
- Digital space representation
  - coding cells, adjacency, incidence
  - oriented cells, boundary operators
- Data structures for subsets of digital space
- Application to digital surface tracking
  - adjacency between boundary elements
  - tracking algorithms
  - benchmarks
- Conclusion and perspectives
Motivation

- Analyzing digital images (2D, 3D, more).
- Writing digital topology and geometry algorithms with application to discrete deformable models.
  - modelling sets of pixels and voxels and their boundaries.
  - tracking digital surfaces; visualizing them.
  - computing geometric characteristics.
Motivation

- Analyzing digital images (2D, 3D, more).
- Writing digital topology and geometry algorithms with application to discrete deformable models.
- digital surface: set of surface elements with topology.
- $r$-cells and sets of $r$-cells in $n$-dimensional space

- How to represent them?
- How to compute their topology: neighborhood, incidences, boundary operators?
- How to get simple geometric characteristics: centroid, normals?
Motivation

- Analyzing digital images (2D, 3D, more).
- Writing **digital topology** and **geometry** algorithms with application to discrete deformable models.
- **digital surface**: set of surface elements with topology.
- \( r \)-**cells** and **sets of \( r \)-cells** in \( n \)-dimensional space

**Objective**: generic answer to digital cell representation.

- independent of space dimension and of cell topology and dimension.
- efficient in practice.
Main objective: analyzing digital images (2D, 3D, more)

⇒ finite regular space of dimension \( n \) and coordinate upper bounds \( M^i \).

\( \text{digital space } \mathbb{C}^n \): cellular decomposition of \( \mathbb{R}^n \) into a regular grid.
Digital space

Main objective: analyzing digital images (2D, 3D, more)
⇒ finite regular space of dimension $n$ and coordinate upper bounds $M^i$.

digital space $\mathbb{C}^n$: cellular decomposition of $\mathbb{R}^n$ into a regular grid.

- good topological properties for surfaces [Kovalevsky89]
- geometric characteristics are always defined.
- many high-level image representation on top of $\mathbb{C}^n$:
  - discrete maps [Braquelaire, Brun, Desbarats, Domenger]
  and [Bertrand, Damiand, Fiorio],
  - cell lists [Kovalevsky],
  - combinatorial pyramids [Brun, Kropatsch].
Usual representations of cells (1/2)

**Cells:**
- pixels, voxels, *spels* in \( n \)D: static arrays of integer
- surfels: pairs of adjacent spels [Herman92]
- other cells: implicitly represented in algorithms

**Set of cells:** characteristic function stored in an “image”; access through offset computation.

⇒ Very simple and easy to implement.

**But** non generic approach
- dimension independence with dynamic allocation (\( \approx 10 \) times slower).
- inhomogeneity between representations of \( n \)-cells, \( n - 1 \)-cells, etc.
Usual representations of cells (2/2)

- **Khalimsky space** $\mathbb{K}^n$: product of $n$ COTS
  - $\mathbb{Z}$ alternating open and closed points: ● × ● × ● × ●
  - $\mathbb{K}^n$ and $\mathbb{C}^n$ are isomorph [Kong,Khalimsky]
  - any $r$-cell: $n$ integer coordinates.
  - cell topology: parity of cell Khalimsky coordinates
  - **Sets of cells**: Characteristic function stored in a doubled “image”. Access through offset computation.
Usual representations of cells (2/2)

- **Khalimsky space** $\mathbb{K}^n$: product of $n$ COTS

  $\Rightarrow$ Homogeneous representation of cells

**But**

- same implementation problems with dynamic arrays
- memory cost of a set of $r$-cells is $2^n \prod (M^i + 1)$ bits.
- signed topology operators (upper and lower boundary) are cumbersome to write.
Proposed representation of cells

- any \( r \)-cell is coded as \textit{one} integer number,
- all the topology (adjacency, incidence) and the geometry (centroid, normal) can be derived from the cell code,
- unoriented and oriented cells can be coded.

\[\Rightarrow\] very compact representation of cells and of sets of cells

- generic: homogeneous representation that is independent of space dimension.
- efficient (e.g. one cpu register stores any cell)
Coding (unoriented) cells

Any \( r \)-cell \( c \) is identified by its Khalimsky coordinates
\[
(x^0_K, \ldots, x^{n-1}_K)
\]
\( c \) is coded as one integer

\[
\alpha \quad x^{n-1} \quad \ldots \quad x^i \quad \ldots \quad x^0
\]

digital coordinate \( x^i = x^i_K \ \text{div} \ 2 \)
each coordinate is binary coded on \( N^i = \log_2(M_i) + 1 \) bits

topology \( \alpha = \sum_i (x^i_K \ \text{mod} \ 2)2^i \)
Coding (unoriented) cells

- Any \( r \)-cell \( c \) is identified by its Khalimsky coordinates \((x^K_0, \ldots, x^n_{K-1})\)

- \( c \) is coded as one integer

Elementary properties

- topology of usual cells: spels (1 \ldots 1), pointels (0 \ldots 0), surfels (1 \ldots 101 \ldots 1).

- 32 bits code any cell of \( 32768 \times 32768 \) 2D image, \( 1024 \times 1024 \times 512 \) 3D image, \( 128^4 \) 4D image. ⇒ enough for most biomedical applications.

- all elementary operations are made with maskings and shiftings, e.g. getting the cell topology or its \( i \)-th coordinate.
Topology operations: adjacency

Two \( r \)-cells with same topology are \( l \)-adjacent iff their coordinates differ of \( \pm 1 \) on \( l \) coordinates.

Cell, 1-adjacent cells, 2-adjacent cells
Topology operations: adjacency

- Two \( r \)-cells with same topology are \( l \)-adjacent iff their coordinates differ of \( \pm 1 \) on \( l \) coordinates.

- Computation of 1-adjacent cells:
  
  \[
  \begin{array}{|c|c|c|c|}
  \hline
  \alpha & x^{n-1} & \ldots & x^i & \ldots & x^0 \\
  \hline
  \end{array}
  \]

  1-adjacent to

  \[
  \begin{array}{|c|c|c|c|}
  \hline
  \alpha & x^{n-1} & \ldots & x^i - 1 & \ldots & x^0 \\
  \hline
  \end{array}
  \]

  \[
  \begin{array}{|c|c|c|c|}
  \hline
  \alpha & x^{n-1} & \ldots & x^i + 1 & \ldots & x^0 \\
  \hline
  \end{array}
  \]

  \( K.\text{adjacent}(c,i,\text{NEG}) \)

  \( K.\text{adjacent}(c,i,\text{POS}) \)
The *low incidence* is the face relation. The 1-low incidence defines the \( r - 1 \)-cells that are faces of a \( r \)-cell.

Cell, 1-low incident cells along \( x \), 1-low incident cells along \( y \)

Prop. Any \( r \)-cell has two 1-low incident \( r - 1 \)-cells along each coordinate where the cell is open.
Topology operations: low incidence

- The *low incidence* is the face relation. The 1-low incidence defines the $r-1$-cells that are faces of a $r$-cell.

- Computation of 1-low incident cells:

  $\cdots 1 \cdots \quad x^{n-1} \quad \cdots \quad x^i \quad \cdots \quad x^0$

  c

  has two 1-low incident cells

  // Space is K

  $\begin{array}{cccc}
  \cdots 0 \cdots & x^{n-1} & \cdots & x^i & \cdots & x^0 \\
  \cdots 0 \cdots & x^{n-1} & \cdots & x^i + 1 & \cdots & x^0 \\
  \end{array}$

  K.lowIncident(c,i,NEG)

  K.lowIncident(c,i,POS)
The *up incidence* is the coface relation. The 1-up incidence defines the $r + 1$-cells that are cofaces of a $r$-cell.

Cell, 1-up incident cells along $x$, 1-up incident cells along $y$

Prop. Any $r$-cell has two 1-up incident $r + 1$-cells along each coordinate where the cell is closed.
The *up incidence* is the coface relation. The 1-up incidence defines the $r + 1$-cells that are cofaces of a $r$-cell.

**Computation of 1-up incident cells:**

<table>
<thead>
<tr>
<th>...0...</th>
<th>$x^{n-1}$</th>
<th>...</th>
<th>$x^i$</th>
<th>...</th>
<th>$x^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

has two 1-up incident cells

<table>
<thead>
<tr>
<th>...1...</th>
<th>$x^{n-1}$</th>
<th>...</th>
<th>$x^i - 1$</th>
<th>...</th>
<th>$x^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ K.\text{upIncident}(c,i,\text{NEG}) \]

\[ K.\text{upIncident}(c,i,\text{POS}) \]
### Cost of elementary cell operations

<table>
<thead>
<tr>
<th>nb ops required</th>
<th>code</th>
<th>topo, coord</th>
<th>==</th>
<th>set coord</th>
<th>adj.</th>
<th>is l-adj.?</th>
<th>inc.</th>
<th>is l-inc.?</th>
</tr>
</thead>
<tbody>
<tr>
<td>bits ops</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>\leq 2n</td>
<td>1</td>
<td>\leq 3</td>
</tr>
<tr>
<td>shifts</td>
<td>(n)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>\leq 6</td>
</tr>
<tr>
<td>integer ops</td>
<td>(n)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>\leq 2n</td>
<td>\leq 1</td>
<td>\leq l + 4</td>
</tr>
<tr>
<td>lut access</td>
<td>(n)</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>\leq n</td>
<td>\leq 2</td>
<td>\leq l + 2</td>
</tr>
<tr>
<td>cond. tests</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>\leq 2n</td>
<td>1</td>
<td>\leq 3l + 1</td>
</tr>
</tbody>
</table>

- The dimension \(n\) is generally low, and \(l \leq n\).
- All these operations on cell codes compete with or are faster than the same operations on cells represented as integer arrays.
Coding oriented cells

- motivation for orienting cells
- useful for defining digital surfaces, cubical cell complexes and boundary operators.
- first step to $r$-dimensional chains
- necessary for giving a local orientation to cells and for defining consistent adjacencies between surfel elements.
motivation for orienting cells

Code of an oriented cell: \[
\begin{array}{cccc}
\alpha & s & x^{n-1} & \ldots & x^i & \ldots & x^0 \\
\end{array}
\]

with orientation \( s \) (0 positive, 1 negative)

\[\rightarrow\] most elementary operations are similar.
Coding oriented cells

- motivation for orienting cells

**Code of an oriented cell:**

\[
\begin{array}{c|c|c|c|c|c|c}
\alpha & s & x^{n-1} & \ldots & x^i & \ldots & x^0 \\
\end{array}
\]

- Boundary operator \( \Delta \): signed 1-low incidence

If \( c = i_k \ldots i_j \ldots i_0 \)

\[
\Delta_{i_j} c = \begin{cases} 
\bar{i}_k \ldots \bar{i}_j \ldots \bar{i}_0 & (-1)^{k-j} s \ x^{n-1} \ \ldots \ x^i \ \ldots \ x^0 \\
\bar{i}_k \ldots \bar{i}_j \ldots \bar{i}_0 & (-1)^{k-j+1} s \ x^{n-1} \ \ldots \ x^i + 1 \ \ldots \ x^0 
\end{cases}
\]

and \( \Delta c = \bigcup_{j=0}^{k} \Delta_{i_j} c \).

- Prop. If \( R \) is a set of oriented \( r \)-cells, then \( \Delta \Delta R = 0 \)

\( \Rightarrow \) Any boundary has no boundary (ie. is closed).
Coding oriented cells

motivation for orienting cells

Code of an oriented cell:

```
| α | s | x^{n-1} | ... | x^i | ... | x^0 |
```

Boundary operator $\Delta$: signed 1-low incidence

Coboundary operator $\nabla$: signed 1-up incidence

Prop. If $R$ is a set of oriented $r$-cells, then $\nabla \nabla R = 0$

$\Rightarrow$ Any coboundary has no coboundary (ie. is open).
Oriented cells and boundaries

**Def.** If a region $O$ is viewed as a set of positively oriented spels, then the boundary of $O$ is $\Delta O$. Any path from $O$ to its complement crosses $\Delta O$.

$$\Rightarrow$$ a simple scanning extracts the boundary of a region.

**Prop.** For any surfel $s \in \Delta O$, $\nabla s = \{+p, -q\}$ with $p \in O$ and $q \not\in O$.

$$\Rightarrow$$ gives locally the inside and outside of a surface.
Data structures for sets in $\mathbb{C}^n$

classical data structures for small sets

<table>
<thead>
<tr>
<th>set data struct.</th>
<th>Is in set ?</th>
<th>Set operations</th>
<th>Memory cost (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dynamic array</td>
<td>$O(m)$</td>
<td>$O(m)+$</td>
<td>$\approx 4m$</td>
</tr>
<tr>
<td>linked list</td>
<td>$O(m)$</td>
<td>$O(m)$</td>
<td>$\approx 24m$</td>
</tr>
<tr>
<td>RB-tree</td>
<td>$O(\log m)+$</td>
<td>$O(\log m)+$</td>
<td>$\approx 3m$</td>
</tr>
<tr>
<td>hashtable</td>
<td>$O(1)+$</td>
<td>$O(1)+$</td>
<td>$\approx 4m'+20m,$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$m' \gg m$</td>
</tr>
</tbody>
</table>

rather costly for big sets: e.g. 1.000.000 surfels $\Rightarrow$ requires 28Mb in a hashtable with $m' = 2m$. 
Data structures for sets in $\mathbb{C}^n$

- classical data structures for small sets
- characteristic function for big sets of $r$-cells.
- code of a cell $c$ gives an offset in array of bits

$$\text{offset}(c) = \text{LUT}[\text{topology}(c)] + \text{coords}(c)$$

\[
\begin{array}{c|c|c}
0 \ldots 0011 \ldots 1 & 0 & 0 \ldots 0 \\
1 \ldots 0101 \ldots 1 & 1 & 0 \ldots 0 \\
1 \ldots 1100 \ldots 0 & \binom{n}{r} - 1 & 0 \ldots 0 \\
\end{array}
\]

- with LUT:

$$\Rightarrow \text{Offset} = \text{one LUT access} + \text{one masking} + \text{one addition}$$
Data structures for sets in $\mathbb{C}^n$

- classical data structures for small sets
- characteristic function for big sets of $r$-cells.

  - set of $r$-cells: $\binom{n}{r} 2^{\sum N_i}$ bits (e.g. set of surfels in a $256^3$ image holds $\approx 50$ million surfels with 6Mb).
  - set of oriented $r$-cells are twice bigger.
  - all set operations are in $O(1)$.
  - computation time of the difference of two sets of surfels in a $512^3$ image: 2.5s or 6ns / surfel (Celeron 450Mhz).
Data structures for sets in $\mathbb{C}^n$

- classical data structures for small sets
- characteristic function for big sets of $r$-cells.
- memory comparison with other cell representations (with space sizes $2^{N_i}$ and $N = \sum N^i$).

<table>
<thead>
<tr>
<th>cell rep.</th>
<th>$r$-cell</th>
<th>set of $n$-cells</th>
<th>set of $r$-cells</th>
<th>genericity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(int. words)</td>
<td>(bits)</td>
<td>(bits)</td>
<td></td>
</tr>
<tr>
<td>classical rep.</td>
<td>$\geq n$</td>
<td>$2^N$</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Khalimsky rep.</td>
<td>$n$</td>
<td>$2^n2^N$</td>
<td>$2^n2^N$</td>
<td>dyn. alloc.</td>
</tr>
<tr>
<td>proposed rep.</td>
<td>1</td>
<td>$2^N$</td>
<td>$(\binom{n}{r})2^N$</td>
<td>yes</td>
</tr>
</tbody>
</table>

Coding cells of multidimensional digital spaces: a framework to write generic digital topology algorithms – p.15/22
Application: digital surface tracking in $\mathbb{C}^n$

Problem: Given an oriented surfel $s$ and a set of spels $O$ with $s \in \Delta O$, find the whole \textit{connected} component $C(\Delta O, s)$ of $\Delta O$ that contains $s$ by tracking \textit{adjacent} element of $\Delta O$.

the algorithm should be linear with respect to the number of surfels of $C(\Delta O, s)$.

an adjacency relation must be defined between surfels of $\Delta O$ (or \textit{bels}).

$2^{n(n-1)/2}$ different \textit{bel adjacencies}.

two of them corresponds to the classical $(2n, 2n^2)$ and $(2n^2, 2n)$ bel adjacencies [Udupa94].
Bel adjacency in $\Delta O$ (1/2)

1. **Def.** A *direct follower* of an oriented $r$-cell $b^r$ is any $r$-cell $c^r \neq b^r$ such that $\exists$ $r-1$-cell $\sigma^{r-1}$ with $+\sigma^{r-1} \in \Delta b^r$ and $-\sigma^{r-1} \in \Delta c^r$. The cell $b^r$ is an *indirect follower* of $c^r$.

2. **Prop.** Any surfel (or $n-1$-cell) $b$ has 3 direct and 3 indirect followers along each coordinate where it is open.

\[ \nabla b = \{ +p, -q \} \]

followers are ordered
Bel adjacency in $\Delta O$ (1/2)

1. **Def.** A *direct follower* of an oriented $r$-cell $b^r$ is any $r$-cell $c^r \neq b^r$ such that $\exists$ $r-1$-cell $\sigma^{r-1}$ with $+\sigma^{r-1} \in \Delta b^r$ and $-\sigma^{r-1} \in \Delta c^r$. The cell $b^r$ is an *indirect follower* of $c^r$.

2. **Prop.** Any surfel (or $n-1$-cell) $b$ has 3 direct and 3 indirect followers along each coordinate where it is open.

3. **Def.** The direct interior (resp. exterior) adjacent bel to $b \in \Delta O$ along coordinate $i$ is the first (resp. last) of the direct followers of $b$ that is $\in \Delta O$. 

![Diagram showing direct and indirect followers of a cell]

Coding cells of multidimensional digital spaces: a framework to write generic digital topology algorithms – p.17/22
Bel adjacency in $\Delta O$ (2/2)

Example of direct interior adjacent bel computation.

Cell Space::DIAdjBel( Set $O$, Cell $b$, int $i$ ) {
    // Extract $p$ and $q$
    int $j$ = orthDir( $b$ );
    bool orth = direct( $b$, $j$ );
    Cell $p$ = unsign( incident( $b$, $j$, orth ) );
    Cell $q$ = adjacent( $p$, $j$, !orth );
    // Extract pp
    bool track = direct( $b$, $i$ );
    Cell pp = adjacent( $p$, $i$, track );
    // Check if first follower $\in \Delta O$
    if ( $O$.isInSet( pp ) )
        return incident( pos( pp ), $i$, track );
    // Extract qq
    Cell qq = adjacent( $q$, $i$, track );
    // Check if second follower $\in \Delta O$
    if ( ! $O$.isInSet( qq ) )
        return adjacent( $b$, $i$, track );
    // if not, last follower $\in \Delta O$
    return incident( neg( qq ), $i$, track );
}
Bel adjacency in $\Delta O$ (2/2)

- Example of direct interior adjacent bel computation.

Prop. Any bel of $\Delta O$ has $n - 1$ direct interior (resp. exterior) adjacent bels in $\Delta O$ and $n - 1$ indirect interior (resp. exterior) adjacent bels.

A bel adjacency relation in $\Delta O$ is given by fixing for each coordinate couple $(i, j)$, $0 \leq i < j < n$, whether the bel adjacency is interior or exterior.

⇒ they are $\frac{n(n-1)}{2}$ different bel adjacencies.
⇒ connectedness relations on $\Delta O$.

From $\Delta \Delta O = 0$ and definition of followers, it is easy to find that tracking along only direct adjacent bels is sufficient to get the whole connected component of $\Delta O$ containing the starting bel.
Digital (hyper)surface tracking

Tracking algorithms

(A) direct and indirect bel adj. tracking
(B) only direct bel adj. tracking
(C) [Herman, Webster83] tracking (3D)

- easily implemented with the proposed framework (both $n$D and 3D algorithms).
Digital (hyper)surface tracking

Set Space::directTracking( Set O, Cell b, BelAdj A ) {
    Set S = emptySurfelSet(); // Output
    Queue Q; // Cells to process
    Q.push( b ); // init tracking
    S.add( b );
    while ( ! Q.empty() ) {
        Cell s = Q.pop(); // current Cell
        for ( i = 0; i < dim(); i++ )
            if ( i != orthDir( s ) ) {
                // Get direct adjacent bel along i
                Cell n = A.directAdj( O, s, i );
                if ( ! S.isInSet( n ) ) {
                    S.add( n );
                    Q.push( n );
                }
            }
    }
    return S;
}
### Benchmarks of boundary extraction algs

#### Experimental results: balls of increasing radius (Celeron 450 Mhz)

<table>
<thead>
<tr>
<th>Space</th>
<th>Rad.</th>
<th>Nb spells</th>
<th>Nb surf.</th>
<th>Scan (A)</th>
<th>Track (A)</th>
<th>Track (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$4096^2$</td>
<td>2000</td>
<td>12566345</td>
<td>16004</td>
<td>2.07s</td>
<td>$&lt; 0.01s$</td>
<td>$&lt; 0.01s$</td>
</tr>
<tr>
<td>$256^3$</td>
<td>30</td>
<td>113081</td>
<td>16926</td>
<td>3.12s</td>
<td>0.02s</td>
<td>0.01s</td>
</tr>
<tr>
<td>$256^3$</td>
<td>60</td>
<td>904089</td>
<td>67734</td>
<td>3.12s</td>
<td>0.09s</td>
<td>0.08s</td>
</tr>
<tr>
<td>$256^3$</td>
<td>120</td>
<td>7236577</td>
<td>271350</td>
<td>3.15s</td>
<td>0.36s</td>
<td>0.32s</td>
</tr>
<tr>
<td>$512^3$</td>
<td>240</td>
<td>57902533</td>
<td>1085502</td>
<td>25.15s</td>
<td>1.88s</td>
<td>1.85s</td>
</tr>
<tr>
<td>$32^4$</td>
<td>14</td>
<td>190121</td>
<td>92104</td>
<td>0.26s</td>
<td>0.14s</td>
<td>0.11s</td>
</tr>
<tr>
<td>$64^4$</td>
<td>14</td>
<td>190121</td>
<td>92104</td>
<td>4.17s</td>
<td>0.20s</td>
<td>0.14s</td>
</tr>
<tr>
<td>$64^4$</td>
<td>30</td>
<td>4000425</td>
<td>904648</td>
<td>4.26s</td>
<td>1.91s</td>
<td>1.37s</td>
</tr>
</tbody>
</table>

- scanning linear with number of surfels of space (in $nD$, 62ns/spel)
- tracking linear with number of bels of boundary (in $nD$, 1.5μs/bel)
Conclusion

- generic framework to represent cells and subsets of digital spaces and to write digital topology algorithms
  - unoriented and oriented $r$-cells
  - compact sets of $r$-cells
  - boundary operators (topology invariants)
  - bel adjacency on boundaries
- proposed framework fully implemented in an object oriented language
  - formal algorithms close to implementation
- tests have shown its efficiency and scalability
check extension of Herman and Webster 3D digital surface algorithm to $n$D spaces
computing $n$-dimensional geometric characteristics
coding cells of hierarchical digital spaces