Implicit Encoding and Simplification/Reduction of nGmaps

Florian Bogner(✉), Jiří Hladůvka, and Walter Kropatsch

Pattern Recognition and Image Processing Group, Vienna University of Technology, Vienna, Austria
`florian.bogner@tuwien.ac.at, {jiri,krw}@prip.tuwien.ac.at`
`https://www.prip.tuwien.ac.at`

Abstract. This paper aims to present a new method of translating labeled 3D scans of biological tissues into Generalized Maps (nGmaps). Creating such nGmaps from labeled images is a solved problem in 2D and 3D using incremental algorithms. We present a new approach that works in arbitrary dimensions. To achieve this in an effective manner, we perform the necessary operations implicitly using theory rather than explicitly in memory. First we define implicit nGmaps. We then present a scheme to construct said nGmap representing an nD pixel/voxel-grid implicitly. Thirdly we give a description of the process needed to reduce such implicit nGmap. We demonstrate that our implicit approach is able to reduce nGmaps in a fraction of otherwise necessary memory.

Keywords: Generalized Maps · nGmaps · Implicit representation · Memory savings

1 Introduction

For analysing CT scans of biological tissues, methods are needed to process the images. Assume that we have a microscopic 3D raster image of tissue and want to run a simulation of physiological processes within, for example leaf tissue and its inherent osmotic movements, respiration and further aspects of biological interest. Assume furthermore that the image is already segmented, meaning each pixel[1] is labeled. This means we know the specific cell or air-pocket a pixel belongs to.

1.1 Problem Statement

For such a simulation we need a data structure where cells and the connections between them are the primary objects. A data structure that meets these requirements and we therefore choose to use, is the n-dimensional Generalized Map (or nGmap for short) [3]. Thus we are faced with the problem of converting the labeled image into an nGmap.

[1] In this paper we use pixel as generic term for any dimension, i.e. including voxels in 3D and hypervoxels in 4D.

É. Baudrier et al. (Eds.): DGMM 2022, LNCS 13493, pp. 110–122, 2022.
https://doi.org/10.1007/978-3-031-19897-7_10

1.2 Prior Work

For 2D-images an algorithm already exists [3]. For 3D-images there is an algorithm for Combinatorial Maps [1]. While this algorithm could be adapted for 3Gmaps, we present a new method that generalizes to arbitrary dimensions.

1.3 Content

To coherently present our approach we first need to explain nGmaps and their specifics. In the following section we give a recap on nGmaps as well as new definitions.

In later sections we will present a new algorithm to translate labeled images into nGmaps. This includes two steps:

1. The implicit construction of the pixel-grid.
2. The contraction of the pixel-grid to adequately represent the labeled regions.

2 Basic Definitions

2.1 nGmap - The Intuitive Definition

An nGmap is a data structure similar to a graph or a mesh. It encodes topological information of a subdivision of an n-dimensional manifold. It consists of so-called i-cells for i from 0 to n. The number i describes the dimension of the i-cell. A 0-cell is a point, a 1-cell is a line bound by two points, i.e. two 0-cells. A 2-cell is a surface patch bound by 1-cells and so on. In general, a $(i + 1)$-cell is bounded by i-cells.

For $i \neq j$ we call an i-cell A *incident* to a j-cell B, if A is in the boundary of B or vice-versa.

Fig. 1. Left: A 2Gmap consisting of five 0-cells, six 1-cells and three 2-cells (including the outside one). Right: The same 2Gmap depicted via its darts, which are drawn as arrows.

These i-cells, however, are not the primary elements used to encode the nGmap, instead so called *darts* are. A dart can be thought of as the intersection of incident i-cells, one for each dimension, i.e. for $i \in \{0, \ldots, n\}$. For example, the marked dart d in Fig. 1 corresponds to the middle-right 0-cell, the top-horizontal 1-cell and the square 2-cell. This interpretation of a dart as an intersection of i-cells is quite important for the intuitive understanding of nGmaps.

For a given i if two darts share the same j-cells for $j \neq i$, but have a different i-cell, we call them *i-linked*. It turns out that a dart is only ever i-linked to a maximum of one other dart. Therefore we can define the involution α_i as the function that maps a dart to its i-linked partner or itself if it has none.

It turns out that the set of darts and the involutions $\alpha_0, \ldots, \alpha_n$ fully describe the structure of the nGmap. Therefore we define an nGmap in the formal definition by its darts and subsequently also define i-cells in terms of darts.

2.2 The Formal Definition

Definition 1. Involution: *A function $f : X \to X$ is called an involution if*

$$\forall x \in X : f(f(x)) = x$$

Definition 2. nGmap: *For $n \in \mathbb{N}_0$ an nGmap or n-dimensional Generalized Map is a tuple $(D, \alpha_0, \ldots, \alpha_n)$, where:*

- *D is a finite set of darts.*
- *For $i \in \{0, \ldots, n\}$ the function $\alpha_i : D \to D$ is an involution.*
- *For $i, j \in \{0, \ldots, n\}, |i - j| \geq 2$ the composition $\alpha_i \circ \alpha_j$ is an involution.*

To define i-cells we first need to define the term orbit:

Definition 3. Orbit: *Let A be a set, $B \subset A$ and $f_1, \ldots, f_n : A \to A$ be functions. Then the orbit of B under $f_1, \ldots, f_n$:*

$$\langle f_1, \ldots, f_n \rangle (B)$$

is the smallest super-set of B closed under $f_1, \ldots, f_n$, i.e.:

- $B \subset \langle f_1, \ldots, f_n \rangle (B) \subset A$
- $\forall i \in \{1, \ldots, n\}, \forall x \in \langle f_1, \ldots, f_n \rangle (B) : f_i(x) \in \langle f_1, \ldots, f_n \rangle (B)$
- *These are all.*

For $x \in A$ the orbit is defined as the orbit of the singleton set $\{x\}$.

For example, the orbit of a single element under a single function is

$$\langle f \rangle (x) = \{x, f(x), f(f(x)), f(f(f(x))), \ldots\}$$

In formal definition i-cells are sets of darts. To a given dart we can find the corresponding i-cell for given i as follows:

Definition 4. i-cell: *Let $(D, \alpha_0, \ldots, \alpha_n)$ be an nGmap, $d \in D$ a dart and $i \in \{0, \ldots, n\}$. The i-cell containing d is defined as the set of darts:*

$$c_i(d) := \langle \alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n \rangle (d)$$

This definition is motivated by the intuitive understanding that α_i changes i-cell and in turn α_j (for $j \neq i$) remains with the same i-cell. Therefore by traversing the orbit of d under α_j for $j \neq i$ we never leave the i-cell and because we consider all α_j, we find every dart of the i-cell.

3 Motivation

3.1 The Naive Algorithm

Let us consider a naive algorithm for the problem:

- Generate a 3Gmap with one cubic 3-cell for every pixel.
- To merge all adjacent pixels with the same label, remove in-between 2-cells.
- Form membranes by merging adjacent 2-cells that border the same 3-cells.
- Form membrane edges by removing every 0-cell that has less than three incident 1-cells.

We now have created a 3Gmap from a labeled image, effectively solving the problem in theory. However, let us estimate the memory requirements: Assuming that a dart is a class consisting of four pointers, one for each involution. A pointer takes up eight bytes in a 64-bit system. A cube in a 3Gmap consists of 48 darts. The 3D-images of the plant scans that motivated this work have a resolution of about 2000^3 pixels. So in total we have

$$2000^3 \times 48 \times 4 \times 8 \text{ bytes} \cong 12 \text{ terabytes}$$

Clearly the memory requirements for creating the pixel-grid mentioned in step one render the naive algorithm infeasible for such a scan. Our solution to circumvent the huge memory requirements is to represent the pixel-grid implicitly, instead of explicitly representing it in memory. Furthermore the reduction as in step 2 onward of the naive algorithm, can be represented implicitly. These two processes will be topic of Sects. 4 and 5 respectively. However one more tool needs to be defined as groundwork before.

3.2 Implicit nGmaps

Definition 5. Implicit nGmap: *For* $n \in \mathbb{N}_0$ *an implicit nGmap is a tuple* $(D, D', \alpha_0, \ldots, \alpha_n)$*, where:*

- *D is a (not necessarily finite) set of darts.*
- *$D' \subset D$ is a finite set of seed-darts.*
- *$\forall i : \alpha_i : D \to D$ is a function. (Not necessarily an involution.)*
- *$(\langle \alpha_0, \ldots, \alpha_n \rangle (D'), \alpha_0, \ldots, \alpha_n)$ is an nGmap, which is called the* Construction.

The idea here is that not all elements of D are darts in the nGmap we want to define. The darts in D' are called seed-darts because from them the orbits grow.

To distinguish, we will also call nGmaps as of Definition 2 *explicit nGmaps*. One major difference between the two types is less of theoretical nature and more related to actual implementations in code:

- Explicit nGmaps can be thought of as being stored in memory, with the α-involutions being implemented via lookup-table or memory pointers. They are mutable.
- Implicit nGmaps however can be thought of as being computed on the fly. Their α-involutions are procedures without state. Therefore they do not occupy much memory, but as a downside they are immutable.

4 Implicit Encoding of the Pixel-Grid

In this section we define an nGmap representing an infinite nD grid. By defining an infinite rather than a finite grid corresponding to the size of the image, we can avoid special cases related to the boundary.

4.1 Darts

As the set of darts we use[2]:

$$D := \mathbb{Z}^n \times \mathbb{N}_{<2^n \cdot n!}$$

A dart is a tuple $d = (p, s) \in D$. The first component $p \in \mathbb{Z}^n$ is called the pixel-position. The second component $s \in \mathbb{N}_{<2^n \cdot n!}$ is called the subpixel-position. Note that there are $2^n \cdot n!$ darts in an nGmap representing a bounded nD-hypercube.

The following is a scheme to enumerate all darts in the interior of an nD cube. Recall that a dart represents the intersection of one i-cell for each i from 0 to n. Thus we describe a dart first by its position via those i-cells and then transform that description into an integer.

4.2 Positional Dart Descriptions

We construct our Positional Dart Description by answering a series of questions.

First: In which n-cell is the dart? We only have one n-cell, so the answer is trivial.

[2] Because the grid is infinite, the construction technically is not an nGmap. One can modify $D := \mathbb{Z}_k^n \times \mathbb{N}_{<2^n \cdot n!}$ using the cyclic group $\mathbb{Z}_k$ for some sufficiently large number k. The nGmap then represents a grid on a large torus and D is finite. When implementing D in code using for example 32-bit ints, this automatically happens with $k = 2^{32}$.

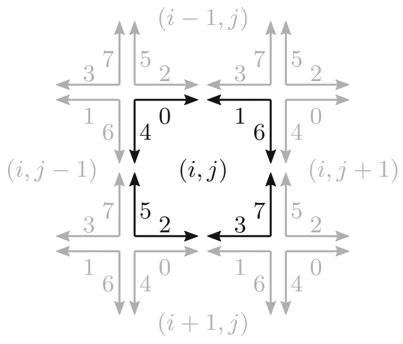

Fig. 2. Representation of one pixel and its neighbors in 2D. (s is attached to each dart, while $p = (i, j)$ is written in the center of each pixel instead of duplicated 8 times.)

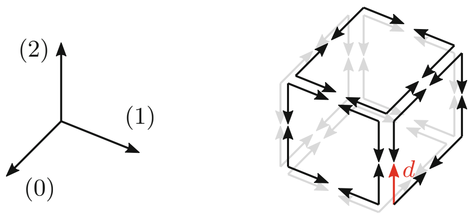

Fig. 3. 3Gmap of a cube, with one dart d marked as an example. Note the orientation and direction of the coordinate axes.

Next: In which $(n-1)$-cell is the dart? There are two $(n-1)$-cells for each coordinate axis, so we can describe the $(n-1)$-cell by the coordinate axis it is perpendicular to and whether it is facing the positive or the negative direction, here called the "top" or the "bottom" respectively. The example dart d in Fig. 3 is in a 2-cell perpendicular to the (1) coordinate axis and on the "top". The descriptions thus begins with:

$$1 \uparrow \dots$$

The identified $(n-1)$-cell now itself consists of $(n-2)$-cells, so our description continues recursively. In which one does it lie?

The example dart d is on the 1-cell perpendicular to the (0) axis and on the "top". Finally, d is on the "bottom" side of the (2) axis. The full description therefore is:

$$1 \uparrow 0 \uparrow 2 \downarrow$$

In general, a description is a list of length $2n$, a permutation of $\{0, \dots, n-1\}$ interleaved with arrows $\downarrow$ or $\uparrow$. As a sanity check, lets calculate the total possibilities: each of the n arrows can be up or down, so we have 2^n possibilities here. The axis numbers can be permuted in $n!$ ways. These are independent, so in total we have $2^n \cdot n!$ possibilities. This exactly matches the number of darts in the nD hypercube.

4.3 Mixed Radix Numbers

To transform a Positional Dart Description into a number, we use a mixed radix numbering system with the signature $(\ldots 5, 2, 4, 2, 3, 2, 2, 2, 1, 2)$ as explained below.

Table 1. Mixed radix system for our dart numbering system

Radix	...	n	2	...	5	2	4	2	3	2	2	2	1	2
Digit worth	...	$2^n n!$	$2^{n-1}n!$	...	768	384	96	48	16	8	4	2	2	1

In a usual numbering system with base b, each digit is worth b times the one on the right. In a mixed radix system, the relative worth of a digit is different for each digit according to its signature. The most common mixed radix system is used to measure time with the signature (7, 24, 60, 60). Each week has 7 d, each day has 24 h, each hour has 60 min, each minute has 60 s.

Table 2. Mixed radix system for time

Name	Weeks	Days	Hours	Minutes	Seconds
Radix	–	7	24	60	60
Digit worth	604800	86400	3600	60	1

To transform a dart description, we translate each part into a digit of the mixed radix. For the arrows we can simply put 0 for ↓ and 1 for ↑.

The axis numbers are not translated directly, i.e. are not the translated digits themselves. Instead, the translation of an axis is its index on the list of not-yet-used axes. This is best explained by example. Let us translate the 4D dart description $0 \uparrow 3 \downarrow 1 \uparrow 2 \uparrow$. At first, no axes were used, so the list is $[0, 1, 2, 3]$. 0 has index 0. Our number thus starts as

$$(01??????)_b$$

The remaining list is now $[1, 2, 3]$ and 3 has index 2 on that list, so the number continues as

$$(0120????)_b$$

The remaining list is now [1,2]. 1 has index 0, and afterward 2 has index 0 so the complete number is

$$(01200101)_b$$

Finally lets translate the number into the decimal system using the 'Digit worth' entries from Table 1.

$$(01200101)_b = 0 \cdot 96 + 1 \cdot 48 + 2 \cdot 16 + 0 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 2 + 1 \cdot 1 = 83$$

Notice how we have for every digit exactly as many choices as is the radix for this digit. Therefore the dart numbers lie flush without gaps.

4.4 Involutions

We define the involutions by lookup-tables (LUT). The LUT maps a subpixel-position to another subpixel-position as well as an offset to the pixel-position. Because only α_n leaves the n-cell i.e. the pixel, the offset is actually only required for this single involution.

We thus define:

$$\alpha_i((p,s)) := \begin{cases} (p, \alpha_i^*(s)) & i < n \\ (p + \Delta p(s), \alpha_i^*(s)) & i = n \end{cases}$$

where α_i^* and Δp are called lookup-tables.

Table 3. Lookup-tables for the 2D case corresponding to Fig. 2 as well as an implementation using bit-flipping magic on the binary representation. (The hat means bit negation.)

s	$\alpha_0^*(s)$	$\alpha_1^*(s)$	$\alpha_2^*(s)$	$\Delta p(s)$
0	1	4	2	(−1, 0)
1	0	6	3	(−1, 0)
2	3	5	0	(1, 0)
3	2	7	1	(1, 0)
4	5	0	6	(0, −1)
5	4	2	7	(0, −1)
6	7	1	4	(0, 1)
7	6	3	5	(0, 1)
abc	$ab\hat{c}$	$\hat{a}cb$	$a\hat{b}c$	N/A

But how can we define these lookup tables? Let us again turn to the Positional Dart Description. First, notation: For $i \in \{0, \ldots, n-1\}$, let $x_i \in \{0, \ldots, n-1\}$ be the axis that the i-cell is perpendicular to and $I_i \in \{\downarrow, \uparrow\}$ be the bottom-top-indicator. $\hat{I}_i$ shall denote the opposite arrow of I_i itself. A general description then looks like this:

$$x_{n-1}I_{n-1} \ldots x_1 I_1 x_0 I_0$$

– α_0^*: The involution α_0 changes 0-cell while staying in the same i-cell for $i > 0$. Thus the start of the description stays the same and only in the last part we swap which side we are on. Thus:

$$\alpha_0^*(x_{n-1}I_{n-1} \ldots x_1 I_1 x_0 I_0) = x_{n-1}I_{n-1} \ldots x_1 I_1 x_0 \hat{I}_0$$

- α_i^* for $0 < i < n$: The involution α_i changes i-cell while staying in the same j-cell for $j \neq i$. Therefore the description before $x_i I_i$ stays the same. The i-cell changes, therefore x_i must change. The original i-cell and the image i-cell intersect in an $(i-1)$-cell. This $(i-1)$-cell is perpendicular to both axes x_i and x_{i-1}. Therefore the image i cell is perpendicular to x_{i-1}. This intersecting $(i-1)$-cell is now on the x_i side of the image i-cell. x_i and x_{i-1} therefore swap places in the description. The arrows swap with them. Afterwards, we are in the same $(i-2)$-cell and so on, so the suffix of the description does not change as well.

$$\alpha_i^*(\dots x_i I_i x_{i-1} I_{i-1} \dots) = \dots x_{i-1} I_{i-1} x_i I_i \dots$$

- α_n^* and Δp: The involution α_n moves us from one n-cell to another, in particular the one that shares the same $(n-1)$-cell. The orientation in regard to the other axes does not change. The direction we move is dependent on I_{n-1}. Therefore we find that:

$$\alpha_n^*(x_{n-1} I_{n-1} \dots x_1 I_1 x_0 I_0) = x_{n-1} \hat{I}_{n-1} \dots x_1 I_1 x_0 I_0$$

$$\Delta p(x_{n-1} I_{n-1} \dots x_1 I_1 x_0 I_0) = \begin{cases} e_{x_{n-1}} & I_{n-1} = \uparrow \\ -e_{x_{n-1}} & I_{n-1} = \downarrow \end{cases}$$

  where e_k is the k-th unit vector.

Note that these definitions elegantly fulfil condition 2 and 3 of Definition 2. Table 3 is generated with these definitions.

4.5 Labels

With the structure of the grid fully defined, we finally need to associate every dart with a label. For a dart $d = (p, s)$ we associate:

- If the pixel-position p is within the image, we associate the label from that pixel in the image.
- Otherwise we associate an additionally created label not occurring in the image called the Out-Of-Bounds-Label. By treating the OOBL as just another label, we can avoid having to consider special cases on the boundary of the image.

Going forward, we denote the set of labels including the OOBL as $\mathbb{L}$ and the association between darts and labels as the function $L : D \to \mathbb{L}$.

5 Implicit Reductions and Contractions

Given an nGmap $(D, \alpha_0, \dots, \alpha_n)$ and a label function $L : D \to \mathbb{L}$ we want to define new involutions β_i and the set D' such that $(D, D', \beta_0, \dots, \beta_n)$ is an implicit nGmap. Note that the original nGmap doesn't have to be the pixel-grid from the previous section. All that is required is that the label function L is *consistent*, meaning all darts from a n-cell map to the same label. We define the β-functions iteratively from the highest dimension to the lowest and then discuss finding an appropriate set of seed-darts D'.

5.1 Defining $\boldsymbol{\beta_n}$

Recall the intuitive understanding of α_n. It changes n-cell while staying in the same i-cell for $i < n$. Since we don't want to remove n-cells, but only merge them later on, we can just define $\beta_n := \alpha_n$.

5.2 Defining $\boldsymbol{\beta_{n-1}}$

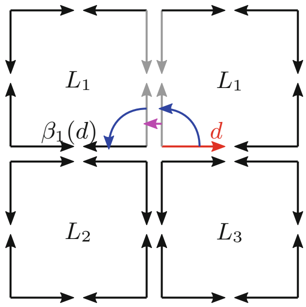

Fig. 4. 2D Example: The grey 1-cell ought to be removed, as it has the same label on both sides. The relevant involutions α_1 and β_2 are illustrated in blue and pink respectively. (Color figure online)

Intuitively the involution α_{n-1} changes to a different $(n-1)$-cell, but such an $(n-1)$-cell may ought to be removed, if the labels on both sides are equal. In this case, illustrated in Fig. 4, we would have to move past that $(n-1)$-cell onto the next one via $\alpha_{n-1} \circ \beta_n \circ \alpha_{n-1}$. Of course, it has to be checked if this next $(n-1)$-cell ought to be removed as well. We define the condition that describes if the $(n-1)$-cell $c_{n-1}(d)$ of a given dart d is removable:

$$R_{n-1}(d) :\Leftrightarrow L(d) = L(\beta_n(d))$$

With this, we can define the new involution:

$$\beta_{n-1}(d) := \begin{cases} \alpha_{n-1}(d) & \\ & \text{if } \neg R_{n-1}(\alpha_{n-1}(d)) \\ \alpha_{n-1} \circ \beta_n \circ \alpha_{n-1}(d) & \\ & \text{elif } \neg R_{n-1}(\alpha_{n-1} \circ \beta_n \circ \alpha_{n-1}(d)) \\ \alpha_{n-1} \circ \beta_n \circ \alpha_{n-1} \circ \beta_n \circ \alpha_{n-1}(d) & \\ & \text{elif } \neg R_{n-1}(\alpha_{n-1} \circ \beta_n \circ \alpha_{n-1} \circ \beta_n \circ \alpha_{n-1}(d)) \\ \dots & \\ (\alpha_{n-1} \circ \beta_n)^k \circ \alpha_{n-1}(d) & \\ & \text{elif } \neg R_{n-1}\left((\alpha_{n-1} \circ \beta_n)^k \circ \alpha_{n-1}(d)\right) \end{cases}$$

The implicit nGmap $(D, D', \alpha_0, \ldots, \alpha_{n-2}, \beta_{n-1}, \beta_n)$ represents an nGmap where the pixels are merged into bigger n-cells according to their labels, but the $(n-1)$-cells are still the sides of a pixel. They can be simplified further.

5.3 Defining β_i for $i \le n-2$

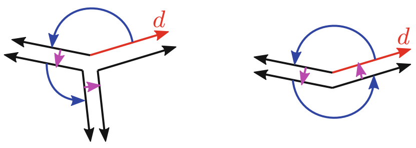

Fig. 5. 2D Example: The left 0-cell is not removable, while the right 0-cell is removable. On the right $\beta_1 \circ \beta_2 \circ \beta_1 \circ \beta_2\,(d)$ loops back, which is not the case on the left.

For the next step, we need to simplify the $(n-2)$-cells, which is accomplished by an appropriate definition of β_{n-2}. Further we also need to simplify the $(n-3)$-cells via β_{n-3} and so on. All of these steps are alike and follow the same definition. For this we need to define when an i-cell is removable. This is only fulfilled if the i-cell has two incident $(i+1)$- cells. [4]

$$R_i\,(d) :\Leftrightarrow \forall d' \in \langle \beta_{i+2}, \ldots \beta_n \rangle\,(d) : d' = \beta_{i+1} \circ \beta_{i+2} \circ \beta_{i+1} \circ \beta_{i+2}\,(d')$$

With this removability criterion illustrated in Fig. 5 we can define:

$$\beta_i\,(d) := \begin{cases} \alpha_i\,(d) & \text{if } \neg R_i\,(\alpha_i\,(d)) \\ \alpha_i \circ \beta_{i+1} \circ \alpha_i\,(d) & \text{elif } \neg R_i\,(\alpha_i \circ \beta_{i+1} \circ \alpha_i\,(d)) \\ \alpha_i \circ \beta_{i+1} \circ \alpha_i \circ \beta_{i+1} \circ \alpha_i\,(d) & \text{elif } \neg R_i\,(\alpha_i \circ \beta_{i+1} \circ \alpha_i \circ \beta_{i+1} \circ \alpha_i\,(d)) \\ \ldots & \\ (\alpha_i \circ \beta_{i+1})^k \circ \alpha_i\,(d) & \text{elif } \neg R_i\left((\alpha_i \circ \beta_{i+1})^k \circ \alpha_i\,(d)\right) \end{cases}$$

The implicit nGmap $(D, D', \alpha_0, \ldots, \alpha_{n-3}, \beta_{n-2}, \beta_{n-1}, \beta_n)$ has simplified n-cells and $(n-1)$-cells, but still has non-simplified i-cells for $i < n-1$. This pattern continues until finally the implicit nGmap $(D, D', \beta_0, \ldots, \beta_n)$ represents the complete reduction of the image.

5.4 Finding Seed-Darts

To build the *Construction* of $(D, D', \beta_0, \ldots, \beta_n)$ we need one seed-dart for each connected component, since the orbit $\langle \beta_0, \ldots, \beta_n \rangle$ of one dart in a connected component cannot reach another connected component. If we know, like in our example-case, the image shows only one connected component, a single seed-dart suffices to create the nGmap for the whole image.

To find a suitable seed-dart we look for a dart d that is not removable at all:

$$\forall i \in \{0, \ldots, n-1\} : \neg R_i\,(d)$$

5.5 Construction

During the *Construction*, i.e. the traversal of the orbit of the seed darts, we create an explicit nGmap. Every encountered implicit dart is associated with an explicit dart. As explicit nGmap we used nGmaps from the CGAL [2]. This explicit nGmap can now be used for further processing, simulations etc.

Optionally, if a bounded nGmap is demanded, the n-cell of the OOBL can be removed. Notice how thanks to the OOBL there were no special cases dealing with the border of the image.

5.6 Limitations

In certain cases the algorithm fails to detect every i-cell. In 3D we found two such cases:

- If a region of one label is completely surrounded by another region, then the 2-cell separating them will erroneously get removed fully. However, this does not occur in plant tissue, as biological cells neither float nor contain each other.
- If two regions are touching and the 2-cell between them is surrounded by a ring-shaped third region, the 2-cell does not get recognized and the first two regions appear disconnected in the resulting 3Gmap. Sadly, this configuration is common in our CT scans.

6 Results

Since the reduction happens implicitly, only the final nGmap needs to be explicitly processed. This means memory is only used for the *Construction* of the final nGmap. This minimizes the necessary memory.

Furthermore, processing only the required minimum of darts allows the algorithm to be fast. For example, a 512^3 labeled image of a leaf cross-section takes about 5 min to be processed on VSC4 (without multi-threading). A 400^3 synthetic image (20^3 checkerboard pattern) takes about 4 min.

In future work, the algorithm needs further refinement to mitigate limitations mentioned in Sect. 5.6. Furthermore, the process could be parallelized to further speed up computations. Finally, in-depth performance profiling and comparison to other approaches should be conducted.

Acknowledgments. This project was supported by the Vienna Science and Technology Fund (WWTF), project LS19-013. The computational results presented have been achieved in part using the Vienna Scientific Cluster (VSC).

References

1. Damiand, G.: Topological model for 3d image representation: definition and incremental extraction algorithm. Comput. Vis. Image Underst. **109**, 260–289 (2008). https://doi.org/10.1016/j.cviu.2007.09.007

2. Damiand, G.: Generalized maps. In: CGAL User and Reference Manual. CGAL Editorial Board, 5.4 (edn.) (2022). https://doc.cgal.org/5.4/Manual/packages.html#PkgGeneralizedMaps
3. Damiand, G., Lienhardt, P.: Combinatorial Maps Efficient Data Structures for Computer Graphics and Image Processing. A K Peters/Crc Press (2014)
4. Illetschko, T.: Minimal combinatorial maps for analyzing 3d data. Technical Report PRIP-TR-110, PRIP, TU Wien (2006). https://www.prip.tuwien.ac.at/pripfiles/trs/tr110.pdf