Implicit Encoding of Contracted Cell nGmaps

Florian Bogner

August 4, 2021

Text colored in blue is new since the last sent draft. Minor changes such as typos and rephrasings are not marked. Footnotes are not meant to go in the final draft. They are ether comments directed directly at Jiří and Walter or jokes to keep me entertained.

Introduction

My idea for creating an implicit encoding of the already contracted and reduced nGmap encoding the topological information in the Leaf-Scans is based on a hierarchy of implicit encodings. In each step a new set of involutions is defined, built from the old ones. Importantly, the involutions are defined as functions instead of stored in tables, thus freeing up memory. Nothing except the labeled image is actually stored in memory.

With these definitions one can then create an actually realized, but small nGmap in memory or maybe continue the use of the implicit encoding. Tests on the trade-off between memory vs. computation have to be done, but I guess memory will be the way to go.

Basic Definitions

To alleviate the confusion between biological cells and nGmap cells, I will call the former bio-cells and the latter 0-cells, 1-cells, etc. or i-cells in general.

As input we are given an segmented image of size $N \times M$ or $N \times M \times K$. Formally we have a set of labels $\mathbb{L}$ (probably subset of the integers) and a function

$$L : \{1, \dots, N\} \times \{1, \dots, M\} \ (\times\{1, \dots, K\}) \to \mathbb{L}$$

This is something I expect to get from Alex.

For compact writing, I define multi-index notation for composition of nGmap operations

$$\alpha_{i,j}(d) := \alpha_j(\alpha_i(d))$$

$$\alpha_{i,j,k}(d) := \alpha_k(\alpha_j(\alpha_i(d)))$$

$$\dots$$

To give an basic overview I will describe the situation in 2D without any edge-cases, such as the edges of the image.

Step 1: Implicit Encoding of the Pixel Grid.

This part is basically done already by Jiří. I have my own ideas, but for now lets assume $\mathbb{D}$ is a set of darts and α_0, α_1 and α_2 are any implicit encoding of the pixel grid.

As part of the definition one has to be able to extract the corresponding pixel coordinates $(i(d), j(d))$ for a given dart d. This will be used to define the Label function on darts:

$$L : \mathbb{D} \to \mathbb{L} : d \mapsto L(i(d), j(d))$$

My Implementation

One way to easily deal with the borders and their special cases is to not have borders at all. We will define a nGmap representing an infinite pixel grid. The set of Darts $\mathbb{D}$ then is infinite, but since we define the Label function to have a certain value for pixels on the outside of the image, the contracted nGmap is finite again.

Darts

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

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

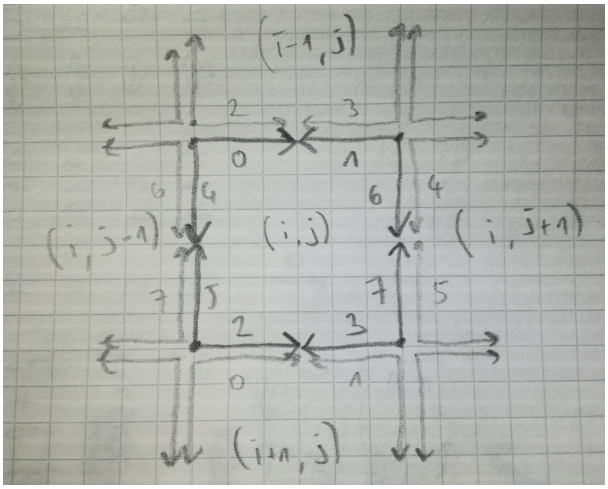

Figure 1: Representation of one pixel and its neighbors in 2D. (s is attached to each Dart, while $p = (i, j)$ is written in the center of each pixel instead of duplicated 8 times.)

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

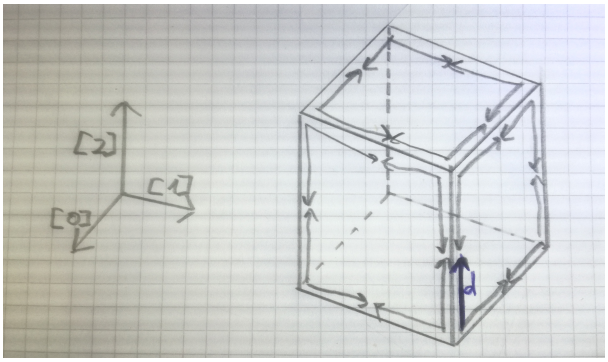

Figure 2: 3-Gmap of a cube, with one dart d marked as an example. Note the orientation and direction of the coordinate axes.

Positional Dart Descriptions

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

Next, we ask: In which $(n-1)$-cell is the dart? There are two $(n-1)$-cell for each coordinate axis, so we can describe the $(n-1)$ by which coordinate axis it is perpendicular to and if it is on the "top" or the "bottom". The example dart d in Figure 2 is in a 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.

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. It doesn't strictly make sense to speak of perpendicularity here, but you get the point. The full description therefore is:

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

In general, a description is a n long list of axis numbers and an arrow $\downarrow$ or $\uparrow$, where axis numbers never repeat. 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, exactly as many darts we have.

Mixed radix numbers

To transform a dart description into a number, we will use a mixed radix numbering system with the signature $(\dots 5, 2, 4, 2, 3, 2, 2, 2, 1, 2)$. What does that mean?

Radix	...	5	2	4	2	3	2	2	2	1	2
Digit worth	...	768	384	96	48	16	8	4	2	2	1

In a usual numbering system with base b, each digits 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 days, each day has 24 hours, each hour has 60 minutes, each minute has 60 seconds.

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. The arrows are easy. $\downarrow$ becomes 0 and $\uparrow$ becomes 1.

The axis numbers are not translated directly, i.e. are not the 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

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

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

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

The 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 with help from the Table above.

$$(01200101)_b = 1 \cdot 1 + 1 \cdot 2 + 2 \cdot 16 + 1 \cdot 48 = 83$$

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

Involutions

The involutions are defined by lookup-tables. The LUT takes a subpixel-position and yields another subpixel-position and an offset to the pixel-position. Because only α_n leaves the n-cell aka. the pixel, the offset is actually only required for this one involution.

We thus define:

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

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

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

Table 1: Lookup-tables for the 2D case corresponding to Figure 1 as well as an implementation using bit-flipping magic on the binary representation. (The hat means bit negation.)

But how can we define these lookup tables? Let us again turn to the positional dart description. First, notation: For $i \in \{0, \dots, n-1\}$, let $x_i \in \{0, \dots, n-1\}$ 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:

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

- 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.[1]

- 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:[2]
$$\alpha_0^*(x_{n-1}I_{n-1}\dots x_1I_1x_0I_0) = x_{n-1}I_{n-1}\dots x_1I_1x_0\hat{I}_0$$
- 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_iI_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}. That 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.[3]
$$\alpha_i^*(\dots x_iI_ix_{i-1}I_{i-1}\dots) = \dots x_{i-1}I_{i-1}x_iI_i\dots$$

Notice how these definitions ~~elegantly~~ fulfil the $\alpha_i \circ \alpha_j = \alpha_j \circ \alpha_i$ for $i + 2 \leq j$ condition. Tabels 1 and 2 are generated with these definitions.

Step 2: Merging Pixels of the same bio-cell.

To merge two pixels, we have to remove the 1-cell between them. Thus, if some α_i reaches it, we go on to some 1-cell that isn't removed.

Thus we define $\beta_0(d) := \alpha_0(d)$ and $\beta_2(d) := \alpha_2(d)$, as these don't leave the 1-cell we are in.

$$\beta_1(d) := \begin{cases} \alpha_1(d) & L(d) \neq L(\alpha_{1,2}(d)) \\ \alpha_{1,2,1}(d) & L(d) = L(\alpha_{1,2}(d)) \neq L(\alpha_{1,2,1,2}(d)) \\ \alpha_{1,2,1,2,1}(d) & L(d) = L(\alpha_{1,2}(d)) = L(\alpha_{1,2,1,2}(d)) \neq L(\alpha_{1,2,1,2,1,2}(d)) \\ \alpha_{1,2,1,2,1,2,1}(d) & \text{else} \end{cases}$$

As we move from 1-cell to 1-cell with α_1 we skip removed 1-cells and corresponding darts until we find a kept 1-cell.

Note that the *else*-case implies all 4 neighboring pixels are of the same label and thus even the starting dart was on a removed edge and in this case $\beta_1(d) = \alpha_2(d)$. For the planned use, the definition of the *else*-case is not strictly necessary, but its good to be complete I guess.

This step has one problem, but it can be fixed: Cells encasing other cells, i.e. shaped like a ring lose the connection as no pseudo-edges are retained. When construction is finished (more on that later) one has to find these lost connections and reinsert the pseudo-edges.

[1] That is why I said we should give up $\alpha_n(d) = d \otimes 10_2$. Notice however, that α_n is still a bitflip, just a different digit and in the mixed radix base.

[2] The fact that this definition satisfies the other wish $\alpha_0(d) = d \oplus 01_2$ is a mere coincidence at this point. It is there because that is where I started my pondering, but I don't explicitly demand that anymore, it just happens. Pretty cool.

[3] This is probably a pretty bad explanation. I hope to do it better in the future. To convince yourself try drawing a cube like in Figure 2, name a dart and name its α_1 and α_2 images. Through examples you will understand better than any explanation. Side Note: I can't believe I am actually giving exercises to my thesis advisors, lol.

Step 3: Simplifying the membranes.

The nGmap described by the $\beta_0, \beta_1, \beta_2$ have one 2-cell correspond to one bio-cell, but many more 1-cells than necessary. We again define a new set of involutions to remove 0-cells bordered by only two 1-cells. We use a similar principle as above to skip such 0-cells.

A 0-cell is removable if for an dart d representing the 0-cell it holds that

$$\beta_{1,2,1,2}(d) = d$$

Note that this is independent of the chosen dart representing the 0-cell. This leads us to the following definition:

$$\gamma_0(d) := \begin{cases} \beta_0(d) & \beta_0(d) \neq \beta_{0,1,2,1,2}(d) \\ \beta_{0,1,0}(d) & \beta_{0,1,0}(d) \neq \beta_{0,1,0,1,2,1,2}(d) \\ \beta_{0,1,0,1,0}(d) & \beta_{0,1,0,1,0}(d) \neq \beta_{0,1,0,1,0,1,2,1,2}(d) \\ \dots & \\ \beta_{0,(1,0)^n}(d) & \beta_{0,(1,0)^n}(d) \neq \beta_{0,(1,0)^n,1,2,1,2}(d) \end{cases}$$

$$\gamma_1(d) := \beta_1(d)$$

$$\gamma_2(d) := \beta_2(d)$$

We again could have a problem here. γ_0 is not an involution for darts that are representing a removable 0-cell. This is not a problem, as these darts are not part of the γ-nGmap anyway. The problem arises, if there is a loop of removable 0-cells in the β-nGmap, for example if L depicts a circle surrounded by a torus. Similar to pseudo-edges we have to insert pseudo-vertices (and probably attach pseudo-edges to them).

Reconstructing the nGmap

Up until now we have only defined the involutions, nothing saved to memory. The set $\mathbb{D}' \subset \mathbb{D}$ of darts that are not removed in the transition from α to β to γ is not explicitly defined. Instead it is found by traversing the γ-nGmap. $\mathbb{D}'$ is significantly smaller than $\mathbb{D}$, so at this point it is[4] worth it to commit it to memory for further processing.

Algorithm 1: Basic Idea of Reconstruction

```
Initialize a set 𝒟 and a queue Q.
Find a dart that is definitely in 𝔻' by picking a dart d' of the edge of the image and
  then d := γ0(d') ∈ 𝔻'.
Insert d into both 𝒟 and Q.
while Q is not empty do
    Pop d from Q
    for i in 0,1,2 do
        d' := γi(d)
        Remember this.
        if d' ∉ 𝒟 then
            Add d' to both 𝒟 and Q
        end
    end
end
```

[4] at least I suspect it is

The "Remember this" step carries a lot of weight, but it ~~is also highly dependent~~ on the underlying data structure, for example CGALs nGmaps or our custom Python ones. Maybe instead of the set $\mathbb{D}$ one should use a hash-map/dict to translate between implicit darts and realized darts of the data structure. This step should also remember an association between 2-cells and labels.

The big problem with this algorithm is already mentioned above, namely it only captures one connected component. There is a way to identify whether we have captured all components or not though. We can compare the complete list of labels $\mathbb{L}$ and the labels reached by the newly constructed nGmap. For any missing labels we can restart the reconstruction at a dart adjacent to a pixel of that missing label and then insert a pseudo-edge. This is repeated until all labels are represented.

In the end, $\mathcal{D} = \mathbb{D}'$ and we have the fully contracted nGmap in memory and can do whatever we want with it.[5]

[5]Save it. Work it. Make it. Do it. Makes us. Harder Better Faster Stronger

s	$\alpha_0^*(s)$	$\alpha_1^*(s)$	$\alpha_2^*(s)$	$\alpha_3^*(s)$	$\Delta p(s)$
0	1	4	16	8	(-1,0,0)
1	0	6	17	9	(-1,0,0)
2	3	5	24	10	(-1,0,0)
3	2	7	25	11	(-1,0,0)
4	5	0	32	12	(-1,0,0)
5	4	2	33	13	(-1,0,0)
6	7	1	40	14	(-1,0,0)
7	6	3	41	15	(-1,0,0)
8	9	12	18	0	(1,0,0)
9	8	14	19	1	(1,0,0)
10	11	13	26	2	(1,0,0)
11	10	15	27	3	(1,0,0)
12	13	8	34	4	(1,0,0)
13	12	10	35	5	(1,0,0)
14	15	9	42	6	(1,0,0)
15	14	11	43	7	(1,0,0)
16	17	20	0	24	(0,-1,0)
17	16	22	1	25	(0,-1,0)
18	19	21	8	26	(0,-1,0)
19	18	23	9	27	(0,-1,0)
20	21	16	36	28	(0,-1,0)
21	20	18	37	29	(0,-1,0)
22	23	17	44	30	(0,-1,0)
23	22	19	45	31	(0,-1,0)
24	25	28	2	16	(0,1,0)
25	24	30	3	17	(0,1,0)
26	27	29	10	18	(0,1,0)
27	26	31	11	19	(0,1,0)
28	29	24	38	20	(0,1,0)
29	28	26	39	21	(0,1,0)
30	31	25	46	22	(0,1,0)
31	30	27	47	23	(0,1,0)
32	33	36	4	40	(0,0,-1)
33	32	38	5	41	(0,0,-1)
34	35	37	12	42	(0,0,-1)
35	34	39	13	43	(0,0,-1)
36	37	32	20	44	(0,0,-1)
37	36	34	21	45	(0,0,-1)
38	39	33	28	46	(0,0,-1)
39	38	35	29	47	(0,0,-1)
40	41	44	6	32	(0,0,1)
41	40	46	7	33	(0,0,1)
42	43	45	14	34	(0,0,1)
43	42	47	15	35	(0,0,1)
44	45	40	22	36	(0,0,1)
45	44	42	23	37	(0,0,1)
46	47	41	30	38	(0,0,1)
47	46	43	31	39	(0,0,1)
$abcdef$	$abcde\hat{f}$	$abcd\hat{f}e$	??	$ab\hat{c}def$	N/A

Table 2: Lookup-tables for a Voxel in 3D. Notice the similarities of the first 8 lines with the 2D table as well as between each block of 8 lines. $\alpha_0(s) = \alpha_0(s \pm 8)$ and $\alpha_1(s) = \alpha_1(s \pm 8)$ as well as $\alpha_3(s) = \alpha_3(s \pm 16)$. Each 8 lines represent a face of the cube, which is in structure similar to a single Pixel in 2D.