Fast algebraic filtering of surfaces from 3D medical images with Julia

Miroslav Jiřík and Alberto Paoluzzi

Abstract

In this paper we introduce a novel algebraic LAR-SURF filter, well founded on algebraic topology methods, to extract and smooth the boundary surface of any subset of voxels arising from the segmentation of a 3D medical image. The input is defined as a *chain*, i.e. as a vector from a linear space of 3-chains, represented in coordinates as a sparse Boolean vector. The output is produced as the result of the mapping via the linear boundary operator $\partial_3 : C_3 \to C_2$ between linear spaces of 3- and 2-chains. In particular, when the input set of voxels is either not (4-)connected, or contains one or more empty regions inside, LAR-SURF generates a non connected set of closed surfaces, i.e. a set of 2-cycles—using the language of algebraic topology. The only data structures used by this approach are sparse arrays with one or two indices, i.e. sparse vectors and matrices. This work is based on LAR (Linear Algebraic Representation) methods, and is implemented in Julia language, natively supporting parallel computing on hybrid architectures.

Contents

1 Introduction

2 Background

2.1 Linear Algebraic Representation

A *representation scheme* for solid modeling is a mapping between a space of mathematical models and a space of symbolic representations, often generated by a formal grammar. Solid pointsets (i.e., 'r-sets') are defined [**?**] as compact (bounded and closed) regular and semianalytic subsets of the d-space. A large number of representation schemes were defined in the past forty years, including the two main classes of (a) *boundary representations* ('B-reps'), where the solid model is represented through a representation of its boundary elements, i.e. faces, edges and vertices, and (b) *decompositive/enumerative representations*, that are a decomposition of either the object or the embedding space, respectively, into a well-defined cellular complex. In particular, a boundary representation provides a cellular decomposition of the object's boundary into cells of dimension zero (vertices), one (edges), and two (faces). Medical imaging is a kind of enumerative representation of organs and tissues of interest, as subsets of 3D volume elements (voxels) from the 3D image.

2.2 Multiindices from Cartesian indices

2.3 Taubin Smoothing

3 Filter design and implementation

3.1 Block-parametric design

3.1.1 Block decomposition

We assume that medical devices produce 3D images with lateral dimensions that are integer multiples of some powers of two, like 128, 256, 512, etc. Any cuboidal portion of image is completely determined by the Cartesian indices of its voxel of lowest and highest indices, and denoted as $Image([\ell_x, \ell_y, \ell_z], [h_x, h_y, h_z])$.

For the sake of simplicity, we assume a common size on the three image axes, and the corresponding image portion B, called *block*, as a function of its element of the lowest block coordinates $i, j, k \in \mathbb{N}$ and of block dimension n:

$$B(i, j, k, n) := Image([in, jn, kn], [i(n+1), j(n+1), k(n+1)])$$

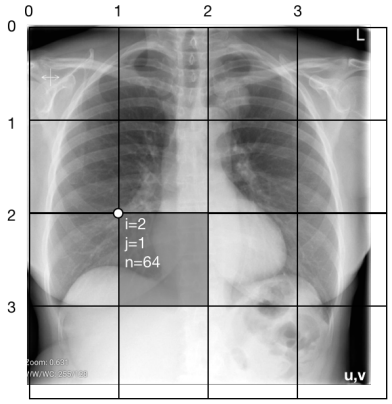

Figure 1: A possible block partitioning of a radiologic image. The evidenced 2D block is $B([2, 1, 64]) = Image([128, 64], [172, 128)]$

Figure 1 shows the block decomposition in a 2D case, with positive integer u, v lateral sizes of image. Note that block sides do not necessarily correspond to image edges.

3.1.2 Block operator

Chain coordinates We are going to treat each image block independently from each other. Hence we map each image subspace $B(i, j, n)$ to the linear *chain* space C_2 of dimension $n \times n$, using coordinate vectors $c_{h,k} \in \mathbb{B}^{n \times n} := \{0, 1\}^{n \times n}$, where the basis element $c = c_{h,k} \in C_2$ is mapped via Cartesian-to-linear map to the Boolean vector

$$Image(h, k) \mapsto c_{h,k} := [0 \cdots 0\ 1\ 0 \cdots 0] \in \mathbb{B}^{n \times n}$$

for each $0 \leq h, k \leq n$, and where the (single) unit element is in position $nk + h \leq n \times n$.

Therefore, each pixel (or voxel) in a block image will be seen as a basis Boolean vector in C_2, and each subset of image elements, as the corresponding Boolean vector in C_2, with many ones as the cardinality of the subset.

Boundary operator For a fixed block size n, the boundary operator $\partial_d : C_d \to C_{d-1}$, with $d \in \{2, 3\}$, will be constructed once and for all using the algorithm given in [], and inlined in the generated boundary extraction code.

It is easy to see that the operator's matrix $[\partial_d]$ is *very sparse*, since it contains $2 \times d$ non-zero elements (ones) for each column (of length n^d), i.e. 4 ones and 6 ones for the 2D and 3D case, respectively. In fact the matrix of a linear operator between linear spaces contains by columns the basis element of the domain space, represented in the target space. In our case, the former is an image element (2-cube or 3-cube), represented as the chain of its boundary—i.e. either a 1-cycle of 4 edges, or a 2-cycle of 6 faces, respectively.

The number of rows of $[\partial_d]$ equates the dimension of the linear space C_{d-1}, i.e. the number of $(d-1)$-cells—elementary $(d-1)$-chains—in the cellular partition of the image. To compute their number, we act in two steps. (a) First we map one-to-one the n^d d-cells with d adjacent $(d-1)$-cells, so getting $d\,n^d$ distinct basis elements of C_{d-1}. (b) Then we complete this bases by adjoining n^{d-1} boundary elements for each of the d dimensions of the image, so providing further $d\,n^{d-1}$ basis elements for C_{d-1}. The dimension of C_{d-1}, and therefore the number of rows of $[\partial_d]$ matrix is $d\,(n^{d-1} + n^d) = d\,n\,(1+n)^{d-1}$. The number of column equates the number of basis elements of C_d, i.e. the number n^d of block elements.

Sparsity and size of boundary matrix As we have seen, we have $2d$ non-zero elements for each column of $[\partial_d]$, so that their total number is $2d\,n^d$. The number of matrix element is $d\,n\,(1+n)^{d-1} \times n^d$, giving a ratio of

$$\frac{\text{non-zero elements}}{\text{total elements}} = \frac{2d \times n^d}{d\,n\,(1+n)^{d-1} \times n^d} = \frac{2}{n + n^d}$$

Using sparse matrices in CSC (Compressed Sparse Column) format we get a storage size:

$$mem([\partial_d]_{n^d}) = 2 \times \#\text{nzero} + \#\text{columns} = 2 \times 2d\,n^d + n^d = (4d+1)n^d.$$

In conclusion, for block size $n = 64$, the matrix $[\partial_d]$ requires for 2D images $9 \times 64^2 = 36,864$ memory elements, and for 3D images $13 \times 64^3 = 3,407,872$ memory elements. Counting the bytes for the standard implementation of a sparse binary matrix (1 byte for values and 8 bytes for indices) we get $(18d+8)n^d$ bytes, giving 176 KB for 2D and 15.872 MB for 3D.

3.1.3 Block boundary mapping

Here we refer directly to the 3D case. Let us call *segment* the bulk content S of interest within the input 3D image of size (u, v, w). We aim to compute the segment boundary $\partial_3 S$. First we set the size n of the block, in order to decompose the input $Image(u, v, w)$ into a fair number

$$M = \lceil u/n \rceil \times \lceil v/n \rceil \times \lceil w/n \rceil$$

of blocks. Then we consider each image portion $c_{i,j,k} = S \cap B(i, j, k, n)$ and compute its (binary) coordinate representation $[c]_{i,j,k} \in C_3(n, n, n)$. This one is a sparse binary vector of length n^3. Then assemble the M representations c of segment portions into a sparse binary matrix $\mathbf{S}$, of dimension $n^d \times M$. Finally compute a matrix $\mathbf{B}$ of boundary portions of S, represented by columns as chain coordinate vectors in C_2:

$$\mathbf{B} = [\partial_3(n, n, n)]\, \mathbf{S}.$$

Of course, the $\mathbf{B}$ sparse matrix has the same column number M of $\mathbf{S}$, because each column contains the boundary representation of the corresponding $S \cap B_{i,j,k}$, and the number of rows of the operator, equal to the dimension of the linear space C_2.

Embedding A final computational step is needed, in order to embed the 2-chains in Euclidean space $\mathbb{E}^3$ and to assemble the whole resulting surface. In particular, we need to compute the *embedding function* $\mu : C_0 \to \mathbb{E}^3$, where C_0 is the space of 0-chains, one-to-one corresponding to the vertices of the extracted surface. The simplest solution is to associate four 0-cells to each 2-cell of the extracted surface, i.e. to each non-zero entry in every column of $\mathbf{B}$. The μ function can be computed by identifying, via element position in the column, a triple of integer values $0 \leq x \leq u, 0 \leq y \leq v, 0 \leq z \leq w$ for each vertex of the 2-cell. The mapping can be implemented using a dictionary, thet will store the inverse coordinate transformation used at the beginning, i.e. the one from linear to Cartesian coords, in order of not duplicating the output vertices.

Assembling the surface portions

3.1.4 Block parallelism

3.1.5 Reduced model

3.2 Julia implementation

3.3 Code optimization

3.4 Performance analysis

4 Examples

5 Conclusion