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

Miroslav Ji**ř**ík 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 Representation Schemes

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 Linear Algebraic Representation

The *Linear Algebraic Representation* (LAR) aims to represent the *chain complex* generated by a piecewise-linear geometric complex embedded either in 2D or in 3D. In few words, it gets a minimal characterization of geometry and topology of a cellular complex, i.e. the embedding mapping $\mu : C_0 \to \mathbb{E}^d$ of 0-cells (vertices), as well a description of $(d-1)$-cells as subsets of vertices, and is able to return the whole chain complex

$$C_\bullet = (C_p, \partial_p) := C_3 \underset{\partial_3}{\overset{\delta_2}{\longleftrightarrow}} C_2 \underset{\partial_2}{\overset{\delta_1}{\longleftrightarrow}} C_1 \underset{\partial_1}{\overset{\delta_0}{\longleftrightarrow}} C_0.$$

and, in particular, any basis for linear chain spaces C_p, and any linear boundary/coboundary map ∂_p and $\delta_p = \partial_{p+1}^\top$ between them. The domain of LAR is the set of chain complexes generated by cell d-complexes ($2 \le d \le 3$). The computer representations of LAR are sparse binary matrices to represent both the operators and the chain bases. Note that in algebraic topology a p-chain is defined as a linear combination of p-cells with scalars from a field. When the scalar coefficients are from $\{-1, 0, +1\}$, a chain may represent *any (oriented) subset of cells* from the cellular complex.

We may therefore get the $(p-1)$-boundary $\partial_p \gamma_p$ of *any* p-chain γ_p, by multiplication of the coordinate representation $[\partial_p]$ of the boundary operator times the coordinate representation $[\gamma_p]$ of the chain in terms of such scalars, i.e. by a matrix-vector product $[\partial_p][\gamma_p]$.

It is easy to understand that the LAR representation scheme is very expressive, i.e. that it has a large domain, including collections of: line segments, quads, triangles, polygons, meshes; pixels, voxels, volume images; B-reps, enumerative and decompositive representations of solids. In this paper we apply LAR methods to computation of boundary representations of solid models from segmentation (labeling) of 3D medical images.

2.2.1 Construction of ∂_d boundary matrix

Once fixed an ordering for all the cells (vertices, edges, pixels, and voxels), i.e. for 0-, 1-, 2-, and 3-elements of a cell partitioning of a 3D image, i.e., once fixed the p-bases for linear spaces C_p of p-chains $(0 \le p \le d)$, we call $M_p = (m_{i,j})$ the *characteristic matrix* of the p-basis, expressed as subsets of 0-cells, where we have that $m_{i,j} = 1$ iff the j-th 0-cell belongs to the i-th p-cell, and $m_{i,j} = 0$ otherwise.

Note that, by computing the (sparse) matrix product $(M_{p-1}M_p^t) = (n_{i,j})$, with $n_{i,j} = \sum_k m_{i,k}m_{k,j}$, we get for each n_{ij} the *number of vertices* shared by c_{p-1}^i and c_p^j. When this number equates the cardinality of c_{p-1}^i, this elementary chain is contained on the boundary of c_p^j. In a 3D image, with cubic 3-cells and squared 2-cells, everywhere we get $n_{i,j} = 4$, we may state $c_2^i \subset \partial c_3^j$. In this case, by looking in each j column of $M_2 M_3^t = (n_{i,j})$, we have exactly *six rows* where $n_{i,j} = 4$.

Finally, consider the linear graded boundary operator $\partial_p : C_p \to C_{p-1}$. As such, it contains by columns the representation of domain basis elements, expressed as linear combination of the basis elements of the range space. Therefore, the operator matrix $[\partial_d]$ is readily obtained by setting $[\partial_d](i,j) = 1$ iff $n_{i,j} = 4$ and $[\partial_d](i,j) = 0$ otherwise. Of course, it will contain six non-zero elements for column. It may be worth to remember that every 3-cell (voxel) has exactly six 2-faces.

It is possible to show that all the interesting relation of incidence/adjacency between cells of different dimensions can be both computed and efficiently queried by pairwise computing some products, with one of terms possibly transposed, of the two boundary and coboundary operator matrices $[\partial_p]$ and $[\delta_p] = [\partial_p^\top] = [\partial_p]^t$. We may also show that such matrices are *very sparse*, with their sparseness growing rapidly with the dimensions (see Section 3.2). The pattern of non-zeros in matrix $[\partial_3]$ corresponding to a brick of shape $(4, 4, 4)$ is given in Figure 1.
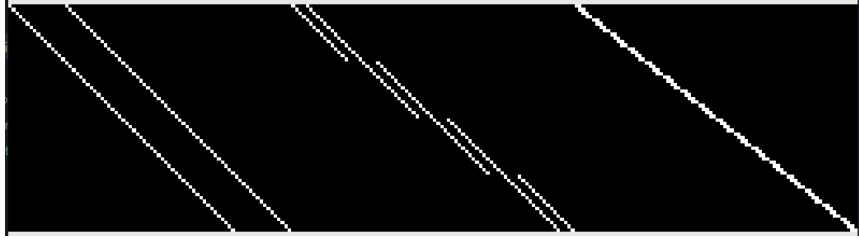

Figure 1: A binary image of the coboundary operator $\delta_2 = \partial^\top : C_2 \to C_3$, built for small 3D images with shape $(4, 4, 4)$. Note that the number of rows equates the cardinality $4 \times 4 \times 4 = 64$ of the voxel set; the number of columns is $d\,n\,(1+n)^{d-1} = 3 \times 4 \times 25 = 300$. Of course, the number of non-zeros per row (cardinality of single voxel facets) is six.

2.3 Multiindices from Cartesian indices

In order to utilize the topological algebra shortly recalled in this paper, we need to explicitly sort the cells of the various dimensions into linearly ordered sequences, possibly according to the linear order their information is linearly accommodated in computer storage.

2.4 Taubin Smoothing

3 Block-parametric design

3.1 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 extracted by multidimensional array *slicing* as $Image([\ell_x{:}h_x, \ell_y{:}h_y, \ell_z{:}h_z])$.

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:

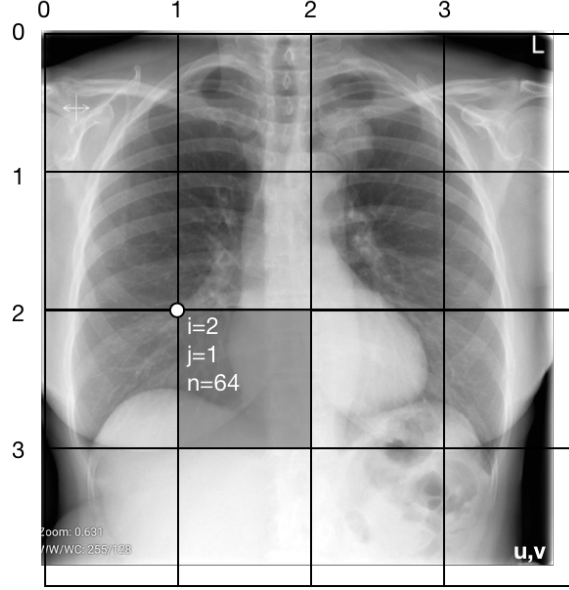$$B(i, j, k, n) := Image([in{:}in + n, jn{:}jn + n, kn{:}kn + n])$$

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

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

3.2 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.

3.3 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

$$M = \lceil u/n \rceil \times \lceil v/n \rceil \times \lceil w/n \rceil = \frac{uvw}{n^3}$$

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:

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

where the boundary matrix has dimension $d\,n\,(1 + n)^{d-1} \times n^d$. 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 $S \cap Box_{i,j,k}i, j, k$, 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 $0 \leq x \leq u$, $0 \leq y \leq v$, and $0 \leq z \leq w$ for each vertex of the 2-cell. The mapping can be implemented using a dictionary, that 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.

Surface assembling All boundary surface subsets $B_{i,j,k}(S) = \partial_3 S \cap Box_{i,j,k}$, provided by columns of $\mathbf{S}$, are embedded in the same coordinate space. In formal terms:

$$\texttt{Lar}(S) := (\texttt{Geom}(S), \texttt{Top}(S)) = (\texttt{V}, \texttt{CV}),$$

where, with respect to the *chain complex* $C_3 \to C_2 \to C_1 \to C_0$ induced by the input image Im and segment $S_{i,j,k}$, we get

$$\texttt{Geom} := \mu(C_0(_{i,j,k})) = \texttt{V}, \tag{1}$$
$$\texttt{Top} := C_3(S) = \mathbf{S} \mapsto \texttt{CV}. \tag{2}$$

and

$$\text{Lar}(B_{i,j,k}) := (\text{Geom}, \text{Top}) = (\text{W}, \text{FW}), \tag{3}$$

$$\text{Geom} := \mu(C_0(B_{i,j,k})) = \text{W} \subset \text{V}, \tag{4}$$

$$\text{Top} := C_2(B_{i,j,k}) = \mathbf{B}_{i,j,k} \mapsto \text{FW} \subset \text{FV}. \tag{5}$$

A translation transformation applied to each vertex subset $\text{W}_{i,j,k}$ with translation vector $\mathbf{t} = [i, j, k]$ will therefore move it in the final space position, so finally giving

$$\text{Lar}(B) = \oplus_{i,j,k}\text{Lar}(\partial_3 S_{i,j,k})) = \oplus_{i,j,k}(\text{W}, \text{FW}).$$

3.4 Block-level parallelism

In the computational pipeline introduced in this paper, several steps can be efficiently performed in parallel at image-block level, depending on the embarassingly data parallel nature of the problem. In particular, little effort is needed to separate the problem into a number of parallel tasks $S_{i,j,k}$, using multiarray slicing. The granularity of parallelism, depending on the block size n, is further enforced by the computation of a single boundary matrix $[\partial_d(n)]$, in turn depending on n, so that the initial communication cost of broadcasting the matrix to nodes can be carefully controlled, and finely tuned depending on the system architecture. The whole approach is appropriate for SIMD hybrid architectures of CPUs and GPUs, since only the initial block setup of boundary matrix and image slices, as well the final collection of computed surface portions, require inter-process communication.

3.5 Julia implementation

3.5.1 Code optimization

3.5.2 Performance analysis

4   Examples

5 Conclusion