
Title:
**Algebraic Filtering of Surfaces from 3D Medical Images with Julia**

Authors:
Miroslav Jirik, jirik@lfp.cuni.cz, Charles University
Alberto Paoluzzi, paoluzzi@dia.uniroma3.it, Roma Tre University

Keywords:
Medical 3D, Computational Topology, Linear Algebraic Representation, LAR, Julia, Surface Extraction

DOI: 10.14733/cadconfP.2020.xxx-yyy

Introduction:

In this paper we introduce a novel algebraic LAR-SURF (Linear Algebraic Representation Surface extraction) 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.

Isosurface extraction to produce geometric models of surfaces from volumetric data is often used for indirect visualization of the medical data or for flow modeling. Here we discuss an approach based on basic algebraic topology and linear algebra, using linear spaces $C_p$ of chains (of cells) of dimension $0 \leq p \leq 3$ and the boundary matrix $[\partial_3] : C_3 \to C_2$.

Input volumetric data are represented by a 3D voxel array and can be generated by segmentation computed tomography (Fig. 2 left) and they are defined as a *chain*, i.e. as a vector from a linear space $C_3$ of 3-chains, represented in coordinates as a sparse binary vector.

A decomposition of the input volumetric data into small submatrices called "bricks" is performed, then the binary coordinate vector of each interesting chain of voxels is generated, and its boundary is computed by matrix multiplication times the boundary matrix producing the binary representation of the boundary surface (surface which define boundary). The output is produced by a linear mapping between spaces of 3- and 2-chains through the boundary operator $\partial_3 : C_3 \to C_2$. 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 triangle faces (Fig. 4).

Parallel data decomposition is used to compute the boundary surface patches within each of the bricks, that are finally joined and smoothed via the Taubin algorithm [9] This work is based on LAR (Linear Algebraic Representation) methods [3, 4] , and is implemented in Julia language, natively supporting parallel computing on hybrid hardware architectures.

Representation Scheme:

With *Boundary representations* ('$B$-reps'), where the solid model is represented through a representation of its boundary elements, i.e. faces, edges and vertices; *decompositive/enumerative representations* [7], 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 can be classified as the *enumerative representation* of cellular decompositions of organs and tissues of interest [5], in particular, as subsets *of 3D volume elements* (voxels) from the 3D image.

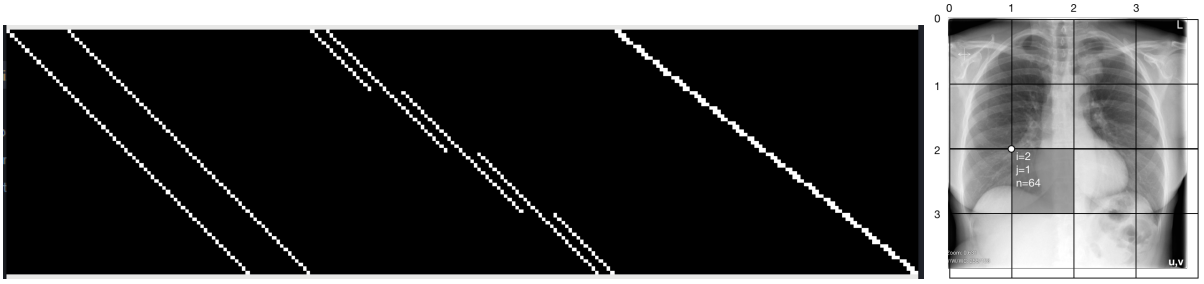


Fig. 1: Left image: the binary image of coboundary matrix built for a small volumetric data/brick with shape $(4, 4, 4)$. The number of rows is $4 \times 4 \times 4$; the number of columns is $d\,n\,(1+n)^{d-1} = 3 \times 4 \times 25$. The right image shows a possible brick partitioning of a radiologic image. The evidenced 2D brick, of size $n^d = 64^2$, is sliced by $\mathbb{B}([2, 1, 64]) = image([128 : 172], [64 : 128))$

LAR aims to represent the *chain complex* [6] generated by a piecewise-linear *geometric complex* embedded either in 2D or in 3D. In a 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

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

and, in particular, any basis for linear chain spaces $C_p$, and any linear boundary/coboundary map $\partial_p$ and $\delta_p = \partial_{p-1}^\top$ between them. The *domain* of LAR is the set of **chain complexes** generated by cell $d$-complexes $(2 \le d \le 3)$. In algebraic topology a $p$-chain is defined as a linear combination of $p$-cells with scalars from a field. We may get the $(p-1)$-boundary $\partial_p c_p$ of *any* $p$-chain $c_p$, by multiplication of the coordinate representation $[\partial_p]$ of the boundary operator times the coordinate representation $[c_p]$ of the chain in terms of such scalars, i.e. by a matrix-vector product $[\partial_p][c_p]$.

The geometry and topology required to display (Fig. 2) a triangulation of boundary faces is contained in the *geometric chain complex* `(geom, top) = (V, (EV, FE, CF))`, where the geometry `geom` is given by the embedding matrix `V` of vertices (0-cells), and topology `top` by the three sparse matrices (`EV`, `FE`, `CF`) of coboundaries $(\delta_0, \delta_1, \delta_2)$ of the chain complex (Fig. 1). The ordered pairs of letters from `V,E,F,C`, correspond to `V`*ertices*→`E`*dges*→`F`*aces*→`C`*ells* into the `C`*olumn*→`R`*ow* order of matrix maps of operators.

Construction of boundary matrix $\partial_d$:

First, let us fix an ordering for all the cells of a partition of input data (with vertices $V$, edges $E$, pixels $F$, and voxels $C$) i.e. for each 0-, 1-, 2-, and 3-cell. These orderings define the $p$-bases for the linear spaces $C_p$ of $p$-chains $(0 \le p \le 3)$. We call $M_p = (m_{i,j})$ the binary *characteristic matrix* of the $p$-basis, expressed as subsets so that $m_{i,j} = 1$ if and only if the $j$-th 0-cell $c_0^j$ belongs to the boundary of $i$-th $p$-cell $m_{i,j}$, and $m_{i,j} = 0$ otherwise.

The product of binary matrices is not binary. In a 3D image, with cubic 3-cells and squared 2-cells in-between, we have exactly *six rows* where $n_{i,j} = 4$, since a cube (3-chain) has six boundary faces (2-chains). The unit incidence coefficients in $[\partial_3]$ are found by filtering with value 4.

Example: Boundary matrices for grids of cubes:

We give here the full Julia code for the algebraic computation of $\partial_3$ matrix, for a very little grid of unit 3-cubes. Due to the simplicity of the cells (voxels = cubes), a sufficient (geom,top) pair is given below as (`V,CV`), where `CV` is an array of arrays of `Float64` indices of grid cubes.

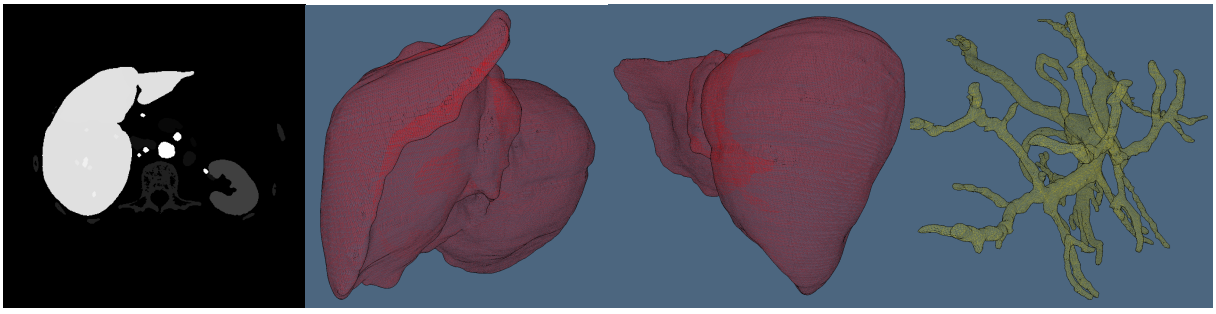

Fig. 2: The left image shows one slice with segmented organs from the Ircad dataset [8]. The other three images show the surface of the liver and portal vein generated by `lar-surf.jl` package.

```
julia> V, CV = Lar.cuboidGrid([3,2,1])
julia> V
0.0 0.0 0.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 2.0 2.0 2.0 2.0 2.0 2.0 3.0 3.0 3.0 3.0 3.0 3.0
0.0 0.0 1.0 1.0 2.0 2.0 0.0 0.0 1.0 1.0 2.0 2.0 0.0 0.0 1.0 1.0 2.0 2.0 0.0 0.0 1.0 1.0 2.0 2.0
0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 1.0
julia> CV
[[ 1, 2, 3, 4, 7, 8, 9,10], [ 3, 4, 5, 6, 9,10,11,12], [ 7, 8, 9,10,13,14,15,16],
 [ 9,10,11,12,15,16,17,18], [13,14,15,16,19,20,21,22], [15,16,17,18,21,22,23,24]
```

Face and Edge Data generation:

In the following, we provide the functions for generating the face data `FV` (vertex indices in faces) with function `CV2FV` and edge data `EV` (vertex indices in edges) with function `CV2EV` from cell data `CV`.

```
function CV2FV( v:: Array { Int64 } )
return faces = [[v[1], v[2], v[3], v[4]], [v[5], v[6], v[7], v[8]],
                [v[1], v[2], v[5], v[6]], [v[3], v[4], v[7], v[8]],
                [v[1], v[3], v[5], v[7]], [v[2], v[4], v[6], v[8]]]
end
function CV2EV( v:: Array { Int64 } )
return edges = [[v[1],v[2]], [v[3],v[4]], [v[5],v[6]], [v[7],v[8]], [v[1],v[3]], [v[2],v[4]],
                [v[5],v[7]], [v[6],v[8]], [v[1],v[5]], [v[2],v[6]], [v[3],v[7]], [v[4],v[8]]]
end
```

Characteristic matrices:

The function `K` transforms an array of arrays ($\mathtt{VV}, \mathtt{EV}, \mathtt{FV}, \mathtt{CV}$) into a sparse binary characteristic matrix ($\mathtt{M}_0, \mathtt{M}_1, \mathtt{M}_2, \mathtt{M}_3$). A Julia sparse matrix needs three arrays I, J, Vals of rows, columns, values of non-zeros:

```
VV = [[v] for v=1:size(V, 2)];
FV = collect(Set{Array{Int64,1}}(cat(map(CV2FV, CV))))
[[13,15,19,21], [1,2,3,4], [7,9,13,15], [13,14,15,16], [7,8,13,14], [1,2,7,8], [2,4,8,10], [7,8,9,10],
 [3,5,9,11], [8,10,14,16], [15,16,21,22], [9,11,15,17], [3,4,5,6], [17,18,23,24], [11,12,17,18],
 [1,3,7,9], [3,4,9,10], [9,10,15,16], [4,6,10,12], [13,14,19,20], [9,10,11,12], [15,16,17,18],
 [19,20,21,22], [15,17,21,23], [16,18,22,24], [21,22,23,24], [10,12,16,18], [5,6,11,12], [14,16,20,22]]

EV = collect(Set{Array{Int64,1}}(cat(map(CV2EV, CV))))
[[15,17], [16,22], [6,12], [17,23], [18,24], [4,10], [3,4], [13,15], [11,12], [9,15], [13,19],
 [1,7], [5,11], [5,6], [12,18], [8,14], [15,21], [17,18], [1,3], [2,4], [16,18], [2,8], [21,23],
 [20,22], [1,2], [14,16], [10,16], [13,14], [19,21], [7,13], [9,10], [23,24], [11,17], [21,22],
 [3,9], [3,5], [9,11], [7,9], [14,20], [7,8], [22,24], [19,20], [8,10], [15,16], [10,12], [4,6]]
```

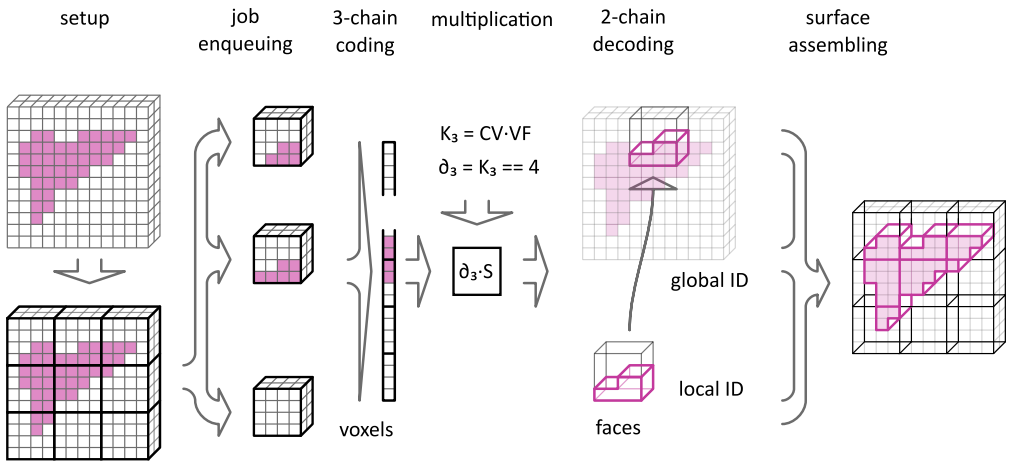


Fig. 3: Workflow of LAR-SURF algorithm

```
function K(CV)
    I = vcat ( [ [k for h in CV[k]] for k =1: length (CV) ]...)
    J = vcat (CV ...)
    Vals = Int8 [1 for k=1: length (I)]
    return SparseArrays . sparse (I,J,Vals)
end
M0 = K(VV); M1 = K(EV); M2 = K(FV); M3 = K(CV)
```

Boundary matrices:

The boundary matrices between non-oriented chain spaces are computed by sparse matrix multiplication followed by matrix filtering, produced in Julia by the broadcast of vectorized integer division ($.\div$):

$\partial_1 = \mathtt{M}_0 * \mathtt{M}_1' = \mathtt{M}_1'$
$\partial_2 = (\mathtt{M}_1 * \mathtt{M}_2') \,.\div \mathtt{sum}(\mathtt{M}_1, \mathtt{dims} = 2)$
$\partial_3 = (\mathtt{M}_2 * \mathtt{M}_3') \,.\div \mathtt{sum}(\mathtt{M}_2, \mathtt{dims} = 2)$

Brick-level parallelism:

Let us assume that medical devices produce 3D images with lateral dimensions that are integer multiples of some powers of two, like 256, 512, etc. Any cuboidal portion of the image $\mathbb{B}$, called *brick*, is completely determined by highest and lowest Cartesian indices of its voxels (Fig. 1 right). We assume $\mathbb{B}$ as a function of its element of the lowest brick coordinates $i, j, k \in [1 : n]$ and the brick lateral size $n \in \mathbb{N}$:

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

The granularity of parallelism, depending on the block size $n$, is further enforced by the computation of a single boundary matrix $[\partial_d(n)]$ depending on $n$.

In our experiment we applied the LAR-SURF filter on 20 segmented volumetric data from dataset Ircad [8] with lateral size $512 \times 512$ and 74 – 260 slices. We compared the performance of LAR-SURF with marching-cubes algorithm implemented in Python. Based on t-test with $\alpha = 0.99$, $p = 8.74 \times 10^{-24}$ and

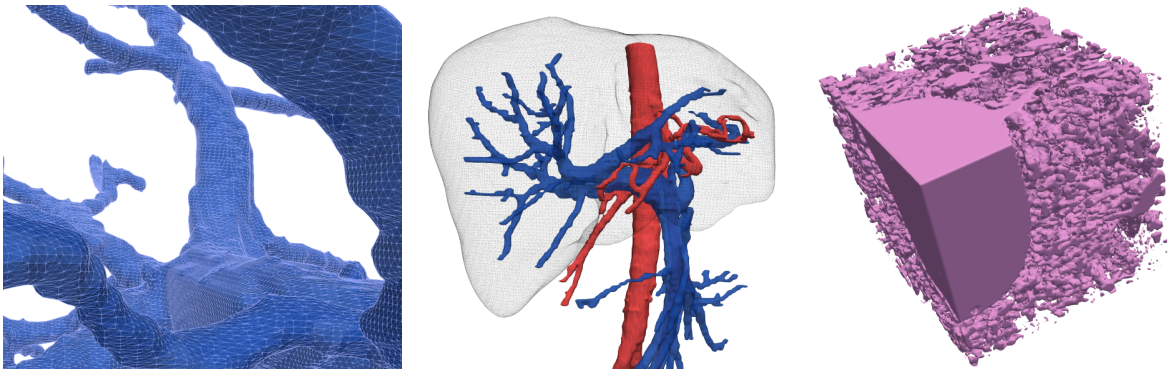

Fig. 4: Images of human liver: (a) portal vein (PV); (b) frontal view, showing also the part of PV connected to the colon and stomach; (c) interlobular veins. Note that brick boundaries look flat.

$s = -16.67$ it can be shown that the mean of time consumed by LAR-SURF is significantly lower from time consumed by marching-cubes.

Discussion of method

The greatest amount of other methods for extraction of boundary surfaces from 3D data arrays—including [10] and [11]—use implicit functions, defined by first averaging upon 3D mesh vertices the lighting or brightness of incident voxels, and then by applying some marching cube algorithm in order to traverse and to triangulate the iso-valued boundary patches.

Conversely, we use a binary labeling of the voxel sets of interested image segment, and the standard medical 3D image array as solid representation. The set of boundary facets is extracted through spMV (sparse matrix vector multiplication) of the $[\partial_3]$ matrix times the binary vector labeling the voxels of the segment. The boundary matrix is computed once and for all, sent to all workers (cores or nodes), then used in parallel for all image brick extractions. This algebraic method can be immediately extended to multi-material processing, as well as to design multi-material tissue layering for 3D printing.

In particular, we might organize a multi-material (or multi-organ) extraction, simply by multiplication of the boundary matrix $[\partial_3] : C_3 \to C_2$ times a binary matrix where the non-zeros elements on each column represents either one of the materials, or one of the organs to be algebraically extracted by the filter. The two surface patches common to two adjacent segments will be exactly coincident, and share the same geometry and topology, with the only exception of each patch contours, where the influence of adjacent patches induces each instance to be separated and round-off.

To make our algorithm consistent with results given by [10] and [11] about the boundaries common to multiple patches is easy: (1) extract the 1-skeleton of the set of patches, using the boundary matrix $[\partial_2] : C_2 \to C_1$ *before* the rounding of surfaces; (2) make the resulting 1-chains curved and smooth, by 1D Taubin algorithm, (3) block their vertices before the Taubin smoothing of incident patches.

The main advantage of the approach discussed in this paper is given by its very algebraic nature; our implementation does not need any kind of algorithmic graph traversal, of difficult parallelization, but only the execution of standard algebraic computing kernels, and in particular the spMV and the spMspM (sparse matrix–sparse matrix multiplication) kernels, nowadays efficiently implemented by tensor processors on Nvidia hardware. As we have shown, the parallelization, both shared-memory and distributed, is also fairly easy and with good speed-up.

Conclusion:

We introduced a Julia implementation of an algebraic filter to extract from 3D medical images the boundary surface of some specific image segment, described as a 3-chain of voxels. Translations from Cartesian indices of cells to linearized indices, the computation of the sparse boundary matrices, and the sparse matrix-vector multiplication are the main computational kernels of this approach.

We may show a good speed-up over marching-cubes algorithms. The existing implementation employs Julia's channels for multiprocessing. Parallelization makes a large portion of spared computational cost. Moreover, we expect additional improvement in the future because our approach is appropriate for SIMD (Single Instruction, Multiple Data) 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.

Currently, the computational pipeline is being strongly improved to gain a greater speed-up using native Julia implementation `CUDA.jl` of Nvidia programming platform [1] , and Julia's `SuiteSparseGraphBLAS.jl` framework [2] for graph algorithms with the language of linear algebra. In particular, we are extending its use pattern in order to work with general cellular complexes.

Acknowledgement:

This work was supported by Charles University Research Centre program UNCE/MED/006 "University Center of Clinical and Experimental Liver Surgery" and Ministry of Education project ITI CZ.02.1.01/0.0/0.0/17_048/0007280: Application of modern technologies in medicine and industry.

References:

[1] Besard, T.; Foket, C.; De Sutter, B.: Effective Extensible Programming: Unleashing Julia on GPUs. IEEE Transactions on Parallel and Distributed Systems, 30(4):827–841, 2019. 10.1109/TPDS.2018.2872064.

[2] Buluc, A.; Mattson, T.; McMillan, S.; Moreira, J.; Yang, C.: Design of the GraphBLAS API for C. Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2017. 2017, 643–652. 10.1109/IPDPSW.2017.117.

[3] Dicarlo, A.; Milicchio, F.; Paoluzzi, A.; Shapiro, V.: Chain-based representations for solid and physical modeling. IEEE Transactions on Automation Science and Engineering, 6(3):454–467, 2009. 10.1109/TASE.2009.2021342.

[4] DiCarlo, A.; Paoluzzi, A.; Shapiro, V.: Linear algebraic representation for topological structures. Computer-Aided Design, 46:269–274, 2014. 10.1016/j.cad.2013.08.044.

[5] Paoluzzi, A.; Dicarlo, A.; Furiani, F.; Jirik, M.: CAD models from medical images using LAR. Computer-Aided Design, 13(6), 2016. 10.1080/16864360.2016.1168216.

[6] Paoluzzi, A.; Shapiro, V.; DiCarlo, A.; Furiani, F.; Martella, G.; Scorzelli, G.: Topological computing of arrangements with (co)chains. Submitted to ACM TSAS, 2017.

[7] Requicha, A. G.: Representations for Rigid Solids: Theory, Methods, and Systems. ACM Comput. Surv., 12(4):437–464, 1980. 10.1145/356827.356833.

[8] Soler, L.: 3D-IRCADb-01 (http://www.ircad.fr/research/3d-ircadb-01/), 2016.

[9] Taubin, G.: A Signal Processing Approach to Fair Surface Design. Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques. ACM, New York, NY, USA, 1995, SIGGRAPH '95, 351–358. 10.1145/218380.218473.

[10] Wang, C. C. L.: Direct Extraction of Surface Meshes from Implicitly Represented Heterogeneous Volumes. Comput. Aided Des., 39(1):35?50, 2007. 10.1016/j.cad.2006.09.003.

[11] Wang, C. C. L.: Extracting Manifold and Feature-Enhanced Mesh Surfaces From Binary Volumes. Journal of Computing and Information Science in Engineering, 8(3), 2008. 10.1115/1.2960489. 031006.