ADDENDUM TO "TOPOLOGICAL COMPUTING OF ARRANGEMENTS WITH (CO)CHAINS"

ALBERTO PAOLUZZI

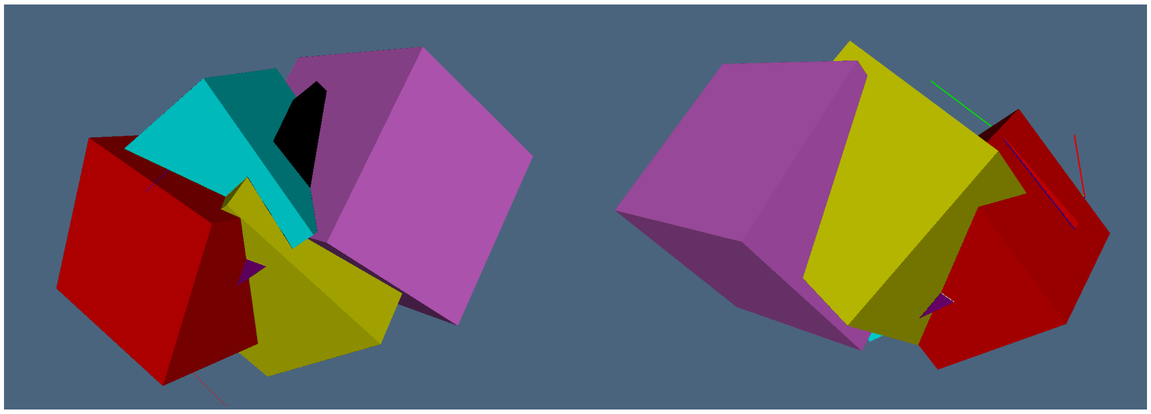

FIGURE 1. Two images of the biggest 3-cell, whose outer 2-boundary coincides with the (reversed) 2-boundary of the outer space (outer 3-cell).

1. Introduction

The aims of this short essay are: (a) to give an example of the computational techniques introduced with the paper "Topological computing of arrangements with (co)chains" [1], submitted about two years ago to ACM Transactions on Spatial Algorithms and Systems (TSAS), and (b) to produce some measures providing a benchmark of this approach.

We intend to illustrate here the input and output data generated by our open-source implementation[1], when applied to a small collection of closed polyhedral surfaces, so producing the *arrangement* (partition) of 3D space, described by the matrices of linear operators $\delta_0, \delta_1, \delta_2$ and their transposed operators $\partial_1, \partial_3, \partial_2$, which go up and down, respectively, between linear chain spaces C_0, C_1, C_2 generated by the 3D partition.

For problem statement, motivation, definitions, discussions, algorithms, applications, and further examples, the reader is referred to [1]. We discuss the Julia's script solving the arrangement problem and the output generated, starting from five randomly oriented and dimensioned cubes around the origin, shown in Figure 1. Finally we compare the amount of data produced by this approach with the crude format of graphics applications nowadays, i.e. the boundary mesh of generated 3-cells.

[1]See our `Github.com` repository,

2. Computation of 3D arrangement example

2.1. Partition into solid cells. A *chain complex* is a short exact sequence of linear spaces C_p of (co)chains, with linear boundary/coboundary maps ∂_p and $\delta_p = \partial_{p+1}^\top$ between:

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

The cells of a space partition are one-to-one with the basis elements of chain spaces. The chain maps (∂_p or δ_p) fully describe the topology of the cellular arrangement.

The set of manifold 3-cells of the 3D arrangement produced by the surfaces in Figure 1 is shown in Figure 2. It is worthwhile to note that such solid cells may be non-convex and non-contractible, i.e., non simply connected. Such properties extend to their 2-cell faces.

It may be interesting to notice also that cells may have different topological genuses, i.e., any number of holes and tunnels. Look for this purpose at the fourth cell (from left) of the second row (from top) of Figure 2. Other 3-cells have also smaller holes in the faces and/or tunnels within. The boundary triangulation of 3-cells needed to generate graphics on a display device or for 3D printing is also provided in our package repository on `github.com`.

The geometric model of the arrangement is generated as a pair (*Geometry*, *Topology*), where the first one is simply given by the embedding map `V` of vertices (0-cells), i.e. by their coordinate matrix, given by columns:

```
julia> V
3x137 Array{Float64,2}:
 1.01181   0.215639  0.516927  0.449016 ... 1.01221   1.30039   0.741732  1.02991
 0.160033  0.06801   0.102833  0.094984 ... 0.816151  0.545669  1.51776   1.24728
 0.196256  0.206963  0.202911  0.203825 ... 0.249344  0.985248  0.613139  1.34904
```

The cardinality of 0-, 1-, 2-, and 3-cell sets `V`, `E`, `F`, `C` are given as row and column numbers of the sparse matrices of their signed incidence relations `EV`, `FE`, `CF`, that can be interpreted as coboundary operators δ_0 (from vertices to edges), δ_1 (from edges to faces), and δ_2 (from faces to solid cells), respectively. The prefix `cop` stands for *chain operator*.

```
julia> copEV              julia> copFE              julia> copCF
268x137 SparseMatrixCSC   157x268 SparseMatrixCSC   27x157 SparseMatrixCSC
with 536 stored entries:  with 786 stored entries:  with 314 stored entries:
  [1  ,   1]  =  -1         [1  ,   1]  =  1          [1  ,   1]  =  -1
  [5  ,   1]  =  -1         [11 ,   1]  =  1          [19 ,   1]  =  1
  [39 ,   1]  =  -1         [2  ,   2]  =  1          [2  ,   2]  =  -1
  ...                       ...                       ...
  [258, 137]  =  1          [156, 267]  =  1          [21 , 156]  =  1
  [260, 137]  =  1          [150, 268]  =  1          [1  , 157]  =  1
  [268, 137]  =  1          [157, 268]  =  1          [21 , 157]  =  -1
```

2.2. The input collection of surfaces. The input data for the example of Figure 1 was generated by a little script in PLaSM language, which is not of interest here. Conversely,

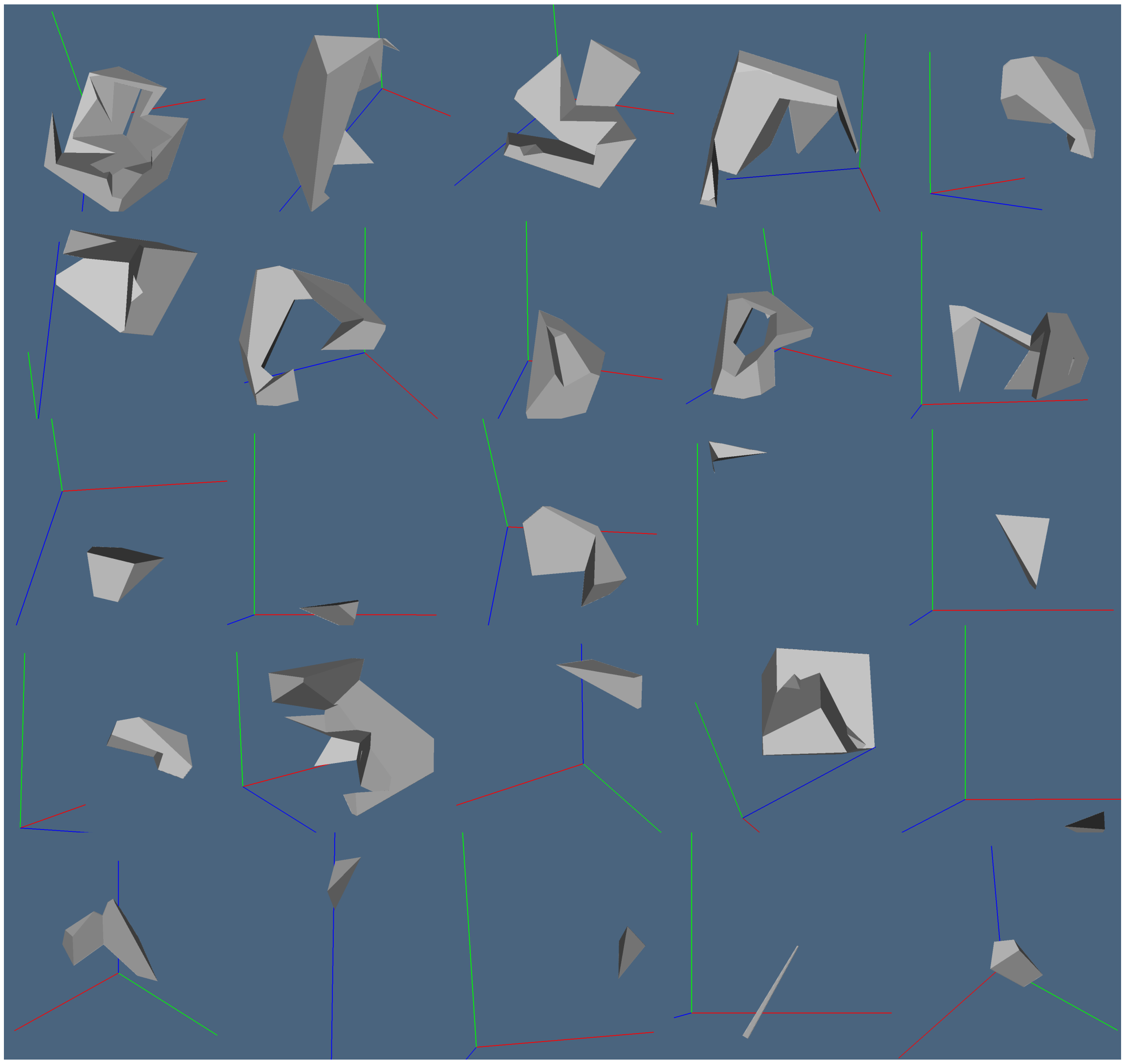

FIGURE 2. The 3-cells of the arrangement of $\mathbb{E}^3$ generated by a collection of five random 3-cubes. Their assembly provides the union of the five 3-cubes. Each 3-cell is given by a column of the sparse matrix of map $\delta_2 : C_2 \to C_3$.

we provide below the full input, i.e. both the geometry `V` and the topology `EV, FV`. Such text data amount to $2.1k$ bytes.

```
julia> @show V;
V = [1.01181 0.215639 0.91979 0.123616 1.02252 0.226347 0.930498 0.134324 0.0458309
-0.301827 0.348275 0.0006172 0.579367 0.23171 0.881811 0.534154 -0.0521776 0.627953
-0.190635 0.489496 -0.0233884 0.656742 -0.161846 0.518285 0.27652 -0.0875132 0.52527
0.161237 0.509324 0.145291 0.758074 0.394041 0.27631 0.564484 0.0058279 0.294002
1.01221 1.30039 0.741732 1.02991; 0.160033 0.0680099 0.956278 0.864255 0.160649
0.0686266 0.956895 0.864872 -0.200245 0.102199 0.417839 0.720283 -0.35354 -0.0510965
0.264543 0.566987 0.682359 0.543901 0.0592036 -0.0792537 0.956374 0.817917 0.333219
0.194762 -0.102028 0.146722 0.324834 0.573584 -0.16916 0.0795901 0.257702 0.506452
0.452356 0.181874 1.15396 0.883481 0.816151 0.545669 1.51776 1.24728; 0.196256
0.206963 0.196872 0.20758 0.997729 1.00844 0.998346 1.00905 0.0677451 0.601282
-0.0855504 0.447986 0.502301 1.03584 0.349005 0.882542 0.159301 0.18809 0.433316
0.462105 0.797002 0.825792 1.07102 1.09981 0.1446 0.377404 0.0774682 0.310272 0.580364
0.813168 0.513232 0.746036 0.403805 1.13971 0.767599 1.5035 0.249344 0.985248 0.613139
1.34904]
julia> @show FV,EV;
(FV,EV) = (Array{Int64,1}[[1,2,3,4],[5,6,7,8],[1,2,5,6],[3,4,7,8],[1,3,5,7],[2,4,6,8],
[9,10,11,12],[13,14,15,16],[9,10,13,14],[11,12,15,16],[9,11,13,15],[10,12,14,16],[17,
18,19,20],[21,22,23,24],[17,18,21,22],[19,20,23,24],[17,19,21,23],[18,20,22,24],[25,26,
27,28],[29,30,31,32],[25,26,29,30],[27,28,31,32],[25,27,29,31],[26,28,30,32],[33,34,35,
36],[37,38,39,40],[33,34,37,38],[35,36,39,40],[33,35,37,39],[34,36,38,40]],
Array{Int64,1}[[1,2],[3,4],[5,6],[7,8],[1,3],[2,4],[5,7],[6,8],[1,5],[2,6],[3,7],[4,8],
[9,10],[11,12],[13,14],[15,16],[9,11],[10,12],[13,15],[14,16],[9,13],[10,14],[11,15],
[12,16],[17,18],[19,20],[21,22],[23,24],[17,19],[18,20],[21,23],[22,24],[17,21],[18,
22],[19,23],[20,24],[25,26],[27,28],[29,30],[31,32],[25,27],[26,28],[29,31],[30,32],
[25,29],[26,30],[27,31],[28,32],[33,34],[35,36],[37,38],[39,40],[33,35],[34,36],[37,
39],[38,40],[33,37],[34,38],[35,39],[36,40]])
```

2.3. **The generating script.** Below we compute the cell 3-complex of the $\mathbb{E}^3$ space partition induced by the above collection of surfaces. In the current prototype implementation, some transformation of input data format is needed. A simpler API will be provided soon. In particular, type `Lar.Cells` (array of arrays of integers) is converted to type `Lar.ChainOp`, i.e., the Julia's type `SparseMatrixCSC{Int8,Int64}` for sparse matrices.

```
cop_EV = Lar.coboundary_0(EV::Lar.Cells);
cop_EW = convert(Lar.ChainOp, cop_EV);
cop_FE = Lar.coboundary_1(V, FV::Lar.Cells, EV::Lar.Cells);
W = convert(Lar.Points, V');

V, copEV, copFE, copCF = Lar.Arrangement.spatial_arrangement(
                         W::Lar.Points, cop_EW::Lar.ChainOp, cop_FE::Lar.ChainOp )
```

2.4. **The output chain complex.** The *geometric chain complex* of our simple example, i.e. the textual values of `V, copEV, copFE`, is given in `github.com/`[2] for a file size of $16k$ bytes, from which `copCF` may be computed, for further $2k$. It may be interesting to note that the minimal and fairly crude `.OBJ` representation of 3-cells, given also in [3], is $15k$ bytes. But whereas this last format just contains a bunch of triangles as triples of indices of vertices, without any storage of the assembly and cells topology, our *chain complex* representation allows for direct answering of either single or batch topological queries through sparse matrix-vector or sparse matrix-matrix multiplication kernels. The

[2]`https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/tree/julia-1.0/examples/3d/data`,
[3]idem

complete representation of both geometry and topology as sets of vertices, edges, faces and 3-cells, is also given in Ref.[4], and weights for $38k$ bytes.

3. Conclusion

In this short *Addendum* to *Topological Computing of Arrangements with (Co)chains*, a small example of computation of a 3D arrangement has been presented and discussed. Our main aim was to show the fairly general nature of generated 3-cells, that are connected and manifold, but non contractible and with any topological genus. We would like to remark also the briefness and simplicity of this representation of *geometric models* as *chain complexes*, and its great generality. This author hopes that the presented material may help to better understand such a novel approach to geometric computing using algebraic topological tools.

References

1. Alberto Paoluzzi, Vadim Shapiro, Antonio DiCarlo, Francesco Furiani, Giulio Martella, and Giorgio Scorzelli, *Topological computing of arrangements with (co)chains*, submitted to Transactions on Spatial Algorithms and Systems, ACM, New York, NY (August 2017).

[4] idem