Homework 2

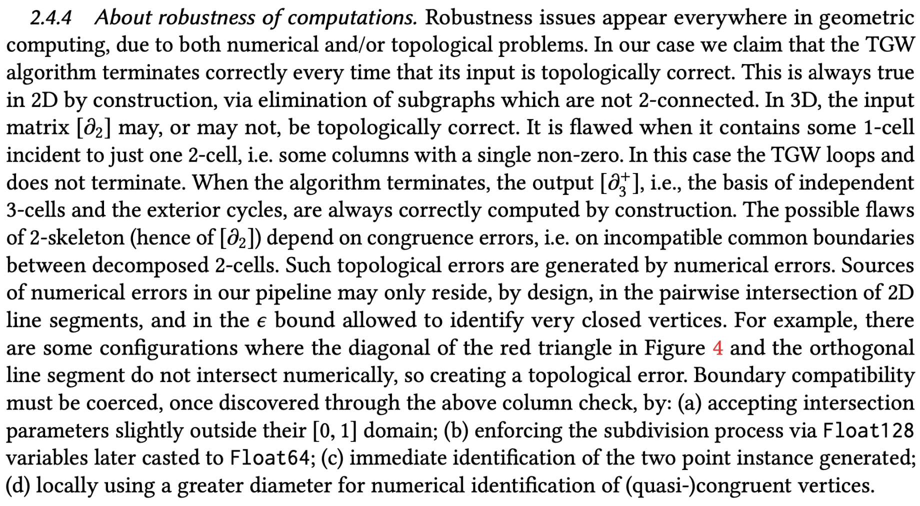
Parallel and Distributed Computation – Roma Tre University

November 27, 2019

Testing Topological and Numerical Robustness

The current implementation of the Julia package LinearAlgebraicRepresentation.jl suffer for lack of sufficient geometric robustness, so that some (many) applications of the algorithmic pipeline for 3D arrangement computation may fail.

About enforcing project robustness



2.4.4 About robustness of computations. Robustness issues appear everywhere in geometric computing, due to both numerical and/or topological problems. In our case we claim that the TGW algorithm terminates correctly every time that its input is topologically correct. This is always true in 2D by construction, via elimination of subgraphs which are not 2-connected. In 3D, the input matrix $[\partial_2]$ may, or may not, be topologically correct. It is flawed when it contains some 1-cell incident to just one 2-cell, i.e. some columns with a single non-zero. In this case the TGW loops and does not terminate. When the algorithm terminates, the output $[\partial_3^+]$, i.e., the basis of independent 3-cells and the exterior cycles, are always correctly computed by construction. The possible flaws of 2-skeleton (hence of $[\partial_2]$) depend on congruence errors, i.e. on incompatible common boundaries between decomposed 2-cells. Such topological errors are generated by numerical errors. Sources of numerical errors in our pipeline may only reside, by design, in the pairwise intersection of 2D line segments, and in the ϵ bound allowed to identify very closed vertices. For example, there are some configurations where the diagonal of the red triangle in Figure 4 and the orthogonal line segment do not intersect numerically, so creating a topological error. Boundary compatibility must be coerced, once discovered through the above column check, by: (a) accepting intersection parameters slightly outside their $[0, 1]$ domain; (b) enforcing the subdivision process via `Float128` variables later casted to `Float64`; (c) immediate identification of the two point instance generated; (d) locally using a greater diameter for numerical identification of (quasi-)congruent vertices.

Figure 1: *From "A Paoluzzi, V Shapiro, A DiCarlo, F Furiani, G Martella, G Scorzelli,* Topological computing of arrangements with (co)chains*"*

Coercing boundary compatibility

Boundary compatibility must be forced by:

(a) accepting intersection parameters slightly outside their $[0, 1]$ domain;

(b) enforcing the subdivision process via `Float128` variables later casted to`Float64`;

(c) immediate identification of the two point instance generated;

(d) locally using a greater diameter for numerical identification of (quasi-)congruent vertices.

The code to rewrite/optimize

Initial version

Study the numeric code inside the file:

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/src/arrangement/planar_arrangement.

In particular, the function `frag_face` and the ones called by it:

```
"""
    frag_face(V, EV, FE, sp_idx, sigma)

`sigma` face fragmentation against faces in `sp_idx[sigma]`
"""
function frag_face(V, EV, FE, sp_idx, sigma)
```

Second version

Study the numeric code inside the file:

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/src/fragface.jl

In particular, the function `fragface` and the ones called by it:

```
"""
    fragface(V::Lar.Points, EV::Lar.Cells, FV::Lar.Cells, FE::Lar.Cells,
        sp_idx::Array, sigma::Int)::Lar.LAR

"""
function fragface(V, cop_EV, cop_FE, sp_idx, sigma)
```

Debug your new code

Debug your new code using the procedures set-up in *Homework 1*.

In particular, write a function testing the existence of $[\partial_2]$ columns having a single non-zero value.

Discuss the results obtained