Computational topology: Lecture 9

Alberto Paoluzzi

April 4, 2019

Introduction

From 2D graph connectivity to $\mathbb{E}^2$ arrangement

Software construction workflow

graph $\longrightarrow$ maximal 2-vertex-connected $\longrightarrow$ 2D arrangement

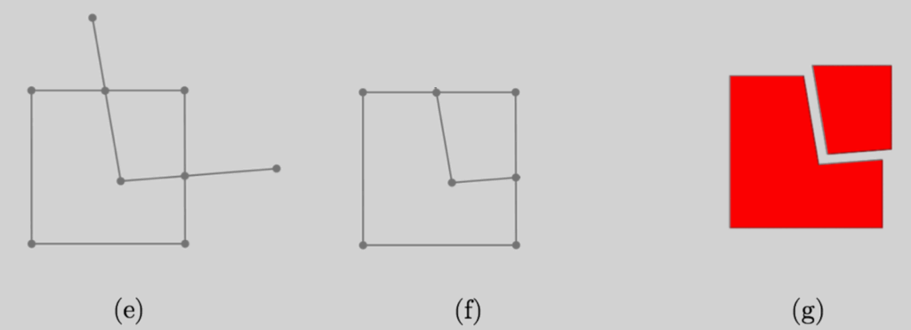

Figure 1: Iterate over 2-cells

Maximal biconnected components

Checks of correctness via graph algorithms

The planar processing of each 2-cell continues by pairwise executing the line segment intersection algorithm, and producing a correct linear graph

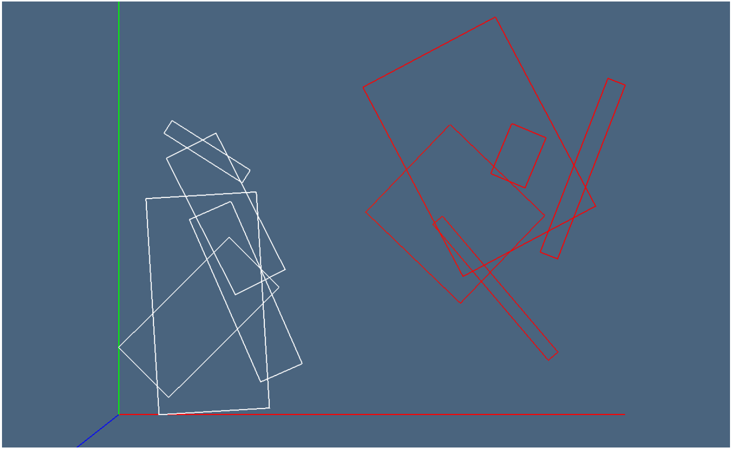

Figure 2: connected components

Removing dangling subcomplexes

- In a d-complex, dangling cells are p-cells, $p < d$, that are not contained in some boundary cycle of a d-cell

Removing dangling subcomplexes

- In a d-complex, dangling cells are p-cells, $p < d$, that are not contained in some boundary cycle of a d-cell

Removing dangling subcomplexes

- In a d-complex, dangling cells are p-cells, $p < d$, that are not contained in some boundary cycle of a d-cell
- In Solid Modeling terminology, they are called non-regular subsets, whence the term regularized Boolean operation

$$A \text{ op}^* \ B = \overline{(A \ \mathring{\text{op}} \ B)}, qquad \text{op} \in \{\cup, \cap, -, \oplus\}$$

Removing dangling subcomplexes

- In a d-complex, dangling cells are p-cells, $p < d$, that are not contained in some boundary cycle of a d-cell
- In Solid Modeling terminology, they are called non-regular subsets, whence the term regularized Boolean operation

$$A \text{ op}^* \ B = \overline{(A \ \mathring{\text{op}} \ B)}, qquad\text{op} \in \{\cup, \cap, -, \oplus\}$$

- Dangling edges are removed using the Hopcroft's and Tarjan's algorithm [1974] for computing the maximal 2-vertex-connected subgraphs

2-vertex-connected graphs

- A connected graph G is 2-vertex-connected if it has at least three vertices and no articulation points

2-vertex-connected graphs

- A connected graph G is 2-vertex-connected if it has at least three vertices and no articulation points
- A vertex is an articulation point if its removal increases the number of connected components of G

2-vertex-connected graphs

- A connected graph G is 2-vertex-connected if it has at least three vertices and no articulation points
- A vertex is an articulation point if its removal increases the number of connected components of G
- Our graphs are also planar by construction

Topological Gift Wrapping

TGW algorithm example 1/6

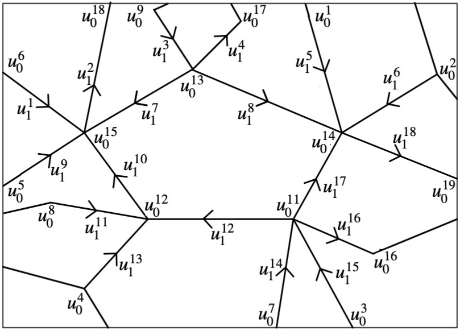

Figure 3: Fragment of $X = X_1$ in $\mathbb{E}_2$, with unit chains $u_0^k \in C_0$ and $u_1^h \in C_1$

We compute stepwise the 1-chain representation $c \in C_1$ of the central 2-cell of the unknown complex $X_2 = \mathcal{A}(X_1)$, using the Topological Gift Wrapping

TGW algorithm example 2/6

Set $c = u_1^{12}$. Then $\partial c = u_0^{12} - u_0^{11}$.

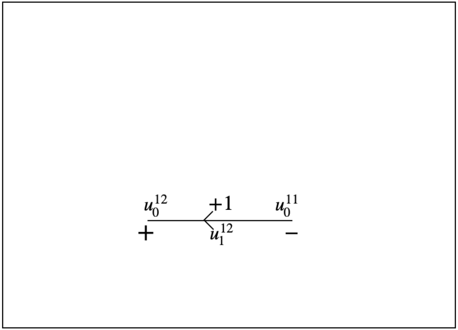

Figure 4: Example step (a)

TGW algorithm example 3/6

$\delta\partial c = \delta u_0^{12} - \delta u_0^{11}$ by linearity. Hence,
$\delta\partial c = (u_1^{10} + u_1^{11} + u_1^{12} + u_1^{13}) - (+u_1^{12} + u_1^{14} + u_1^{15} + u_1^{16} + u_1^{17})$.

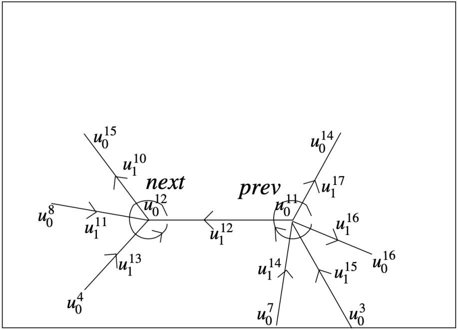

Figure 5: Example step (b)

TGW algorithm example 4/6

By computing corolla(c), we get

$$\begin{aligned}
&\text{corolla}(c) \\
&= c + \text{next}(c \cap \delta\partial c) \\
&= c + \text{next}(u_1^{12})(\delta u_0^{12}) - \text{next}(u_1^{12})(\delta u_0^{11}) \\
&= u_1^{12} + \text{next}(u_1^{12})(\delta u_0^{12}) + \text{prev}(u_1^{12})(\delta u_0^{11}) \\
&= u_1^{12} + u_1^{10} + u_1^{17}.
\end{aligned}$$

If c is coherently orientd, then

$$c = u_1^{10} + u_1^{12} - u_1^{17}$$

and

$$\partial c = u_0^{15} - u_0^{12} + u_0^{12} - u_0^{11} + u_0^{11} - u_0^{14} = u_0^{15} - u_0^{14}.$$

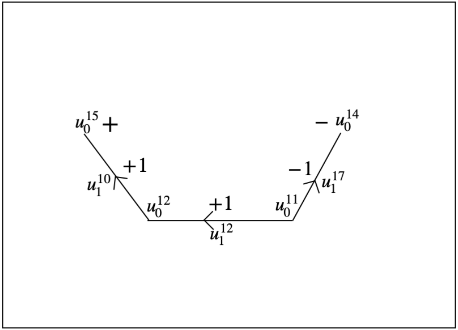

Figure 6: Example step (c)

TGW algorithm example 5/6

Repeating and orienting coherently the computed 1-chain yields:

$$
\begin{aligned}
&\mathrm{corolla}(c)\\
&= c + \mathrm{next}(c \cap \delta\partial c)\\
&= c + \mathrm{next}(u_1^{10})(\delta u_0^{15}) - \mathrm{next}(u_1^{17})(\delta u_0^{14})\\
&= u_1^{10} + u_1^{12} - u_1^{17} + \mathrm{next}(u_1^{10})(\delta u_0^{15}) +\\
&\quad + \mathrm{prev}(u_1^{17})(\delta u_0^{14})\\
&= u_1^{10} + u_1^{12} - u_1^{17} - u_1^{7} + u_1^{8}
\end{aligned}
$$

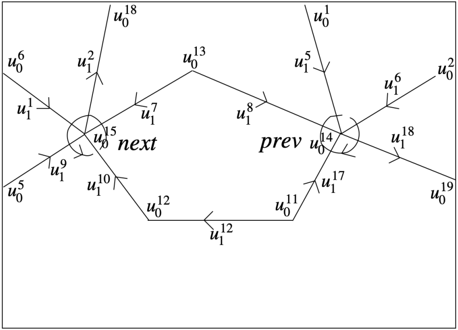

Figure 7: Example step (d)

TGW algorithm example 6/6

$$\partial \text{corolla}(c) = \emptyset$$

and the extraction algorithm terminates, giving

$$c = u_1^{10} + u_1^{12} - u_1^{17} - u_1^{7} + u_1^{8}$$

as the $C_1(X)$ representation of a basis element of $C_2(X)$, with $X = \mathcal{A}(X_1)$.

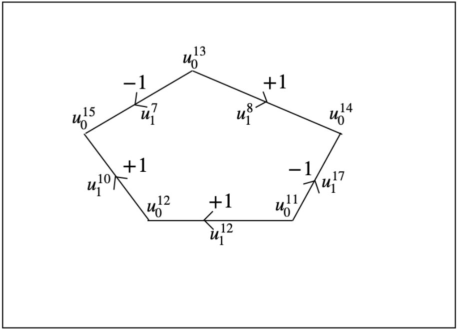

Figure 8: Example step (e)

The coordinate vector of this cycle is accommodated as a new signed column of the yet partially unknown sparse matrix $[\partial_2]$ of the operator $\partial_2 : C_2 \to C_1$.

TGW example

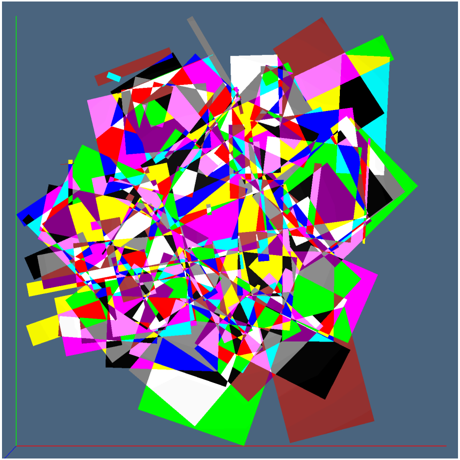

Figure 9: Example

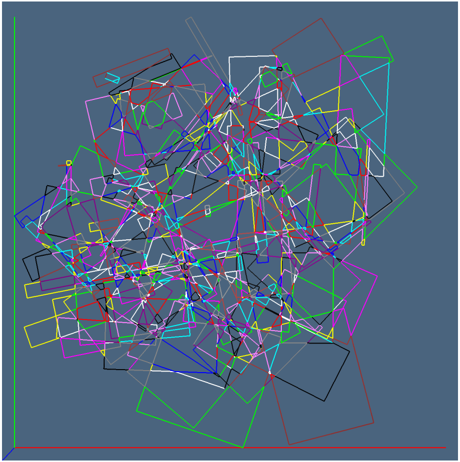

Figure 10: Example

TGW example

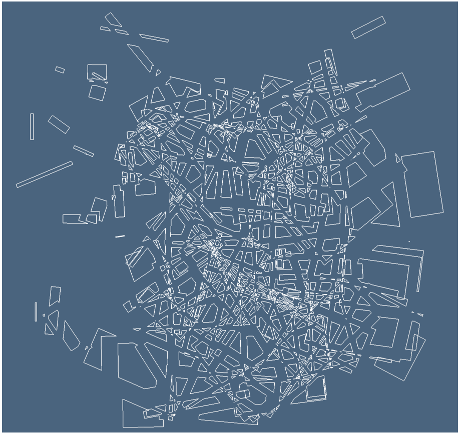

Figure 11: Example

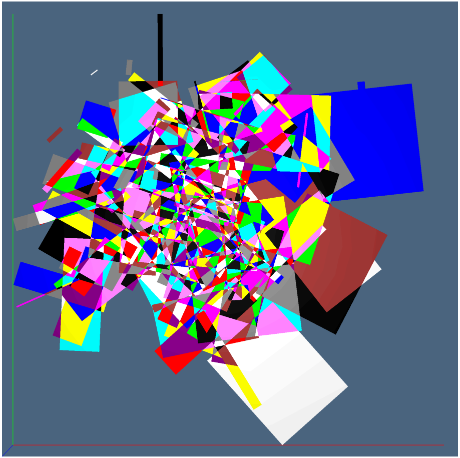

Figure 12: Example

TGW 2D script 1/2 (example)

```
# preliminaries
using LinearAlgebraicRepresentation
Lar = LinearAlgebraicRepresentation
using Plasm

# random data generation
V,EV = Lar.randomcuboids(100, .4)
V = Plasm.normalize(V,flag=true)
Plasm.view(Plasm.numbering(.05)((V,[[[k] for k=1:size(V,2)], EV])))

# 2D arrangement
W = convert(Lar.Points, V')
cop_EV = Lar.coboundary_0(EV::Lar.Cells)
cop_EW = convert(Lar.ChainOp, cop_EV)
V, copEV, copFE =
    Lar.Arrangement.planar_arrangement(W::Lar.Points, cop_EW::Lar.ChainOp)
```

TGW 2D script 2/2 (visualization)

```
# 2-cell random colors
triangulated_faces = Lar.triangulate2D(V, [copEV, copFE])
V = convert(Lar.Points, V')
FVs = convert(Array{Lar.Cells}, triangulated_faces)
Plasm.viewlarcolor(V::Lar.Points, FVs::Array{Lar.Cells})

# 1-cell random colors
EVs = Lar.FV2EVs(copEV, copFE) # polygonal face fragments
Plasm.viewlarcolor(V::Lar.Points, EVs::Array{Lar.Cells})

# exploded polygons
model = V,EVs
Plasm.view(Plasm.lar_exploded(model)(1.2,1.2,1.2))
```

TGW pseudocode

TGW pseudocode

ALGORITHM 2: *Computation of signed $[\partial_d^+]$ matrix*

/* **Pre-condition:** d equals the space $\mathbb{E}^d$ dimension, such that $(d-1)$-cells are shared by *two* d-cells */
/* */
Input: $[\partial_{d-1}]$ # Compressed Sparse Column (CSC) signed matrix (a_{ij}), where $a_{ij} \in \{-1, 0, 1\}$
Output: $[\partial_d^+]$ # CSC signed matrix of cycles
$[\partial_d^+] = []$; $m, n = [\partial_{d-1}].shape$; $marks = Zeros(n)$ # initializations
while $\mathrm{Sum}(marks) < 2n$ **do**
  $\sigma = Choose(marks)$ # select the $(d-1)$-cell seed of the column extraction
  if $marks[\sigma] == 0$ **then** $[c_{d-1}] = [\sigma]$
  else if $marks[\sigma] == 1$ **then** $[c_{d-1}] = [-\sigma]$
  $[c_{d-2}] = [\partial_{d-1}][c_{d-1}]$ # compute boundary c_{d-2} of seed cell
  while $[c_{d-2}] \neq []$ **do** # loop until boundary becomes empty
    $corolla = []$
    for $\tau \in c_{d-2}$ **do** # for each "hinge" τ cell
      $[b_{d-1}] = [\tau]^t[\partial_{d-1}]$ # compute the τ coboundary
      $pivot = \{|b_{d-1}|\} \cap \{|c_{d-1}|\}$ # compute the τ support
      if $\tau > 0$ **then** $adj = Next(pivot, Ord(b_{d-1}))$ # compute the new adj cell
      else if $\tau < 0$ **then** $adj = Prev(pivot, Ord(b_{d-1}))$
      if $\partial_{d-1}[\tau, adj] \neq \partial_{d-1}[\tau, pivot]$ **then** $corolla[adj] = c_{d-1}[pivot]$ # orient adj
      else $corolla[adj] = -(c_{d-1}[pivot])$
    end
    $[c_{d-1}]$ += $corolla$ # insert *corolla* cells in current c_{d-1}
    $[c_{d-2}] = [\partial_{d-1}][c_{d-1}]$ # compute again the boundary of c_{d-1}
  end
  for $\sigma \in c_{d-1}$ **do** $marks[\sigma]$ += 1 # update the counters of used cells
  $[\partial_d^+]$ += $[c_{d-1}]$ # append a new column to $[\partial_d^+]$
end
return $[\partial_d^+]$