Computational topology: Lecture 8

Alberto Paoluzzi

March 28, 2019

Problem: compute the $\mathbb{E}^2$ partition

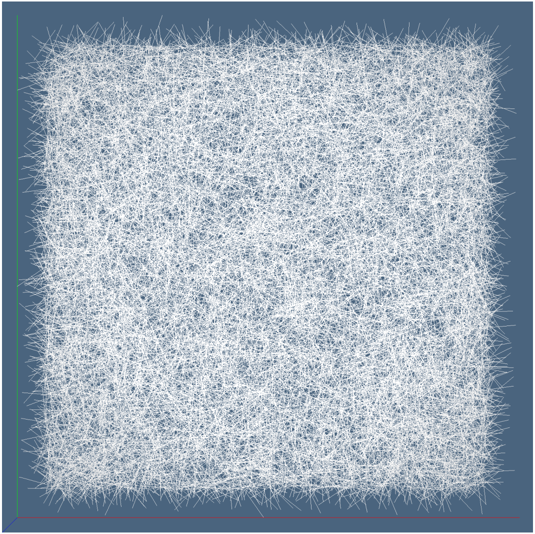


Figure 1: A set of line segments in normalized space $[0, 1]^2$

Problem: compute the $\mathbb{E}^2$ partition

1. Reduction of arrangements to segment intersection
2. Segment intersection
3. Planar graph by congruence
4. Maximal biconnected components

Reduction of arrangements to segment intersection

Problem statement

Input: collection $\mathcal{S}$ of piecewise-linear geometric complexes of dimension $(d-1)$, embedded in $\mathbb{E}^d$ space, with $d \in \{2, 3\}$

Problem statement

Input: collection $\mathcal{S}$ of piecewise-linear geometric complexes of dimension $(d-1)$, embedded in $\mathbb{E}^d$ space, with $d \in \{2, 3\}$

Examples include soups of lines or polygons, triangled surfaces, quads from cubical meshes, 1-, 2-, or 3-cells from 2D or 3D image elements, i.e.pixels or voxels, 2-skeletons/boundaries of triangulated polyhedra, non manifold B-reps or decompositive reps of solid models

Problem statement

Input: collection $\mathcal{S}$ of piecewise-linear geometric complexes of dimension $(d-1)$, embedded in $\mathbb{E}^d$ space, with $d \in \{2, 3\}$

Examples
: include soups of lines or polygons, triangled surfaces, quads from cubical meshes, 1-, 2-, or 3-cells from 2D or 3D image elements, i.e.pixels or voxels, 2-skeletons/boundaries of triangulated polyhedra, non manifold B-reps or decompositive reps of solid models

Geometric complex
: pair (X, μ), where (a) X is a cellular complex specifying the topology (b) $\mu : X_0 \to \mathbb{E}^d$ is an embedding function of 0-cells (geometry)

Problem statement

Input: collection $\mathcal{S}$ of piecewise-linear geometric complexes of dimension $(d-1)$, embedded in $\mathbb{E}^d$ space, with $d \in \{2, 3\}$

Examples include soups of lines or polygons, triangled surfaces, quads from cubical meshes, 1-, 2-, or 3-cells from 2D or 3D image elements, i.e.pixels or voxels, 2-skeletons/boundaries of triangulated polyhedra, non manifold B-reps or decompositive reps of solid models

Geometric complex pair (X, μ), where (a) X is a cellular complex specifying the topology (b) $\mu : X_0 \to \mathbb{E}^d$ is an embedding function of 0-cells (geometry)

Problem statement

Input: collection $\mathcal{S}$ of piecewise-linear geometric complexes of dimension $(d-1)$, embedded in $\mathbb{E}^d$ space, with $d \in \{2, 3\}$

Examples include soups of lines or polygons, triangled surfaces, quads from cubical meshes, 1-, 2-, or 3-cells from 2D or 3D image elements, i.e.pixels or voxels, 2-skeletons/boundaries of triangulated polyhedra, non manifold B-reps or decompositive reps of solid models

Geometric complex pair (X, μ), where (a) X is a cellular complex specifying the topology (b) $\mu : X_0 \to \mathbb{E}^d$ is an embedding function of 0-cells (geometry)

The data may contains both $(d-1)$- and d-complexes: the combinatorial union of their $(d-1)$-skeletons is selected as the actual input to the pipeline

Problem statement

Input: collection $\mathcal{S}$ of piecewise-linear geometric complexes of dimension $(d-1)$, embedded in $\mathbb{E}^d$ space, with $d \in \{2, 3\}$

Examples include soups of lines or polygons, triangled surfaces, quads from cubical meshes, 1-, 2-, or 3-cells from 2D or 3D image elements, i.e.pixels or voxels, 2-skeletons/boundaries of triangulated polyhedra, non manifold B-reps or decompositive reps of solid models

Geometric complex pair (X, μ), where (a) X is a cellular complex specifying the topology (b) $\mu : X_0 \to \mathbb{E}^d$ is an embedding function of 0-cells (geometry)

The data may contains both $(d-1)$- and d-complexes: the combinatorial union of their $(d-1)$-skeletons is selected as the actual input to the pipeline

Definition

An input collection $\mathcal{S}$ of geometric complexes that mutually intersect, will partition $\mathbb{E}^d$ into a cellular complex $X = \bigcup X_p$ $(0 \leq p \leq d)$, called the arrangement $\mathcal{A}(\mathcal{S})$ induced by $\mathcal{S}$

Computational pipeline

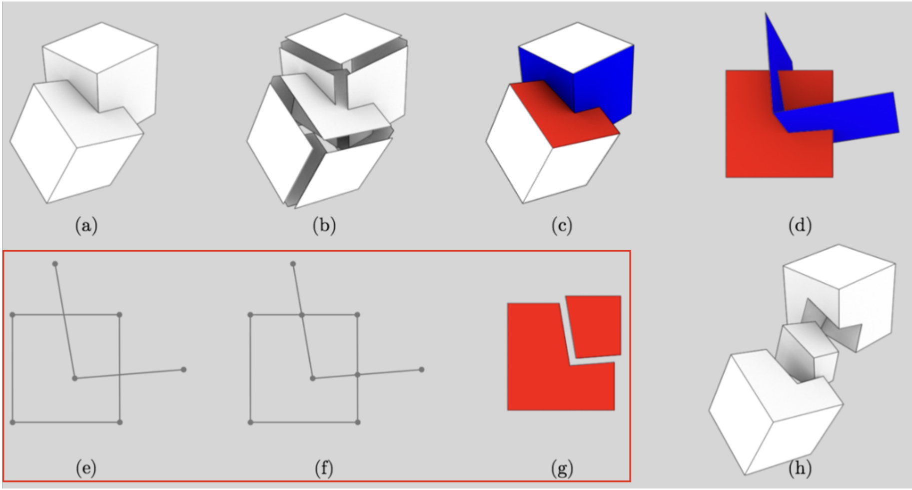


Figure 2: Pipeline segment in 2D

From segment intersection to $\mathbb{E}^2$ arrangement

Software construction workflow

testing segment intersection 2D arrangement

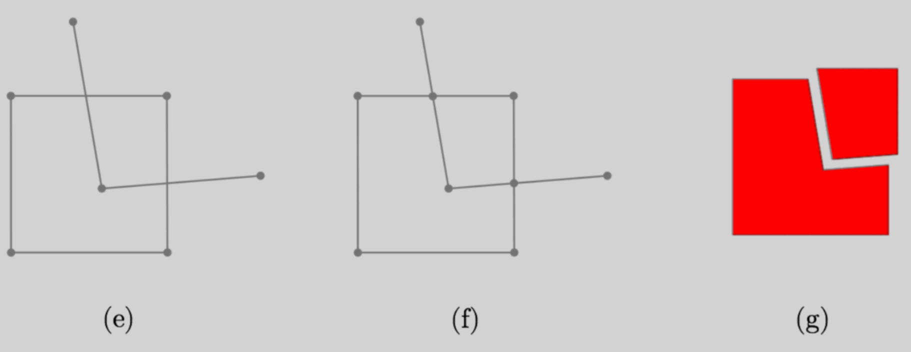


Figure 3: Iterate over 2-cells

Correctness and robustness

- The computation is correct and robust because the boundaries of adjacent decomposed 2-cells are compatible as cellular complexes by construction

Correctness and robustness

- The computation is correct and robust because the boundaries of adjacent decomposed 2-cells are compatible as cellular complexes by construction
- This fact is induced by continuity of topological spaces, mathematically modeled here by chains of cells of a complex

Correctness and robustness

- The computation is correct and robust because the boundaries of adjacent decomposed 2-cells are compatible as cellular complexes by construction
- This fact is induced by continuity of topological spaces, mathematically modeled here by chains of cells of a complex
- A requirement of the standard definition of a cellular complex demands boundary compatibility to hold

Correctness and robustness

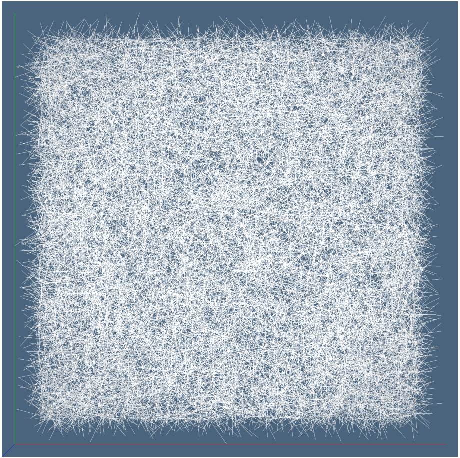
- The computation is correct and robust because the boundaries of adjacent decomposed 2-cells are compatible as cellular complexes by construction
- This fact is induced by continuity of topological spaces, mathematically modeled here by chains of cells of a complex
- A requirement of the standard definition of a cellular complex demands boundary compatibility to hold
- This fact is guaranteed here, since abutting subsets of 1-cells have non-empty intersection, so they generate congruent 0-, 1-cells

Segment intersection

Test-driven development

Always start by parametrically generating some test data

In our case start looking to a script file in `/repo/examples`



PARAMETRIC !! generation of random test data

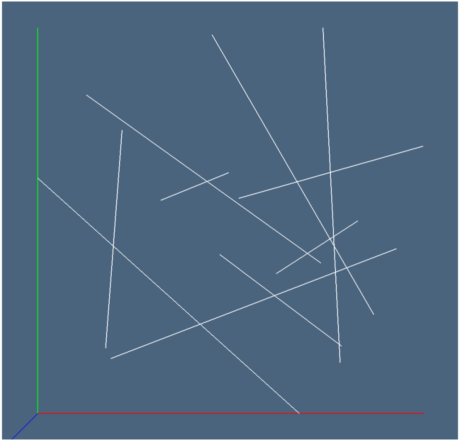


Figure 5: Some examples

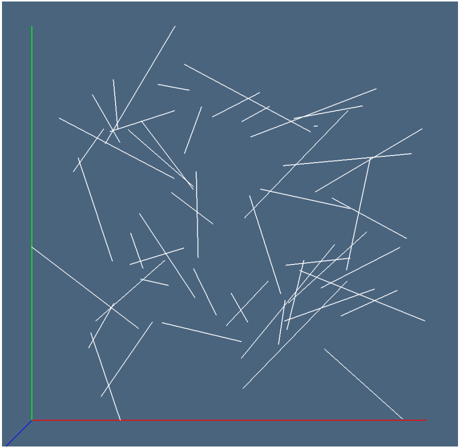


Figure 6: Some examples

Code: examples/randomlines.jl

```
using LinearAlgebraicRepresentation
Lar = LinearAlgebraicRepresentation
using Plasm

n = 50 #1000 #1000 #20000
t = 0.5 #0.15 #0.4 #0.15
V = zeros(Float64,2,2*n)
EV = [zeros(Int64,2) for k=1:n]

for k=1:n
    v1 = rand(Float64,2)
    v2 = rand(Float64,2)
    vm = (v1+v2)/2
    transl = rand(Float64,2)
    V[:,k] = (v1-vm)*t + transl
    V[:,n+k] = (v2-vm)*t + transl
    EV[k] = [k,n+k]
end

V = Plasm.normalize(V)
model = (V,EV)
Plasm.view(model)
```

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/ 1.0/src/refactoring.jl

```
function fragmentlines(model)
    V,EV = model

    # acceleration via spatial index computation
    # actual parametric intersection of each line with the clo
    # initialization of local data structures
    # generation of intersection points
    # normalization of output

    return V,EV
end
```

Acceleration via spatial index computation

Sigma = Lar.spaceindex(model)

```
function spaceindex(model::Lar.LAR)::Lar.Cells
    V,CV = model[1:2]
    dim = size(V,1)

    cellpoints = [ V[:,CV[k]]::Lar.Points
        for k=1:length(CV) ]
    bboxes = [hcat(Lar.boundingbox(cell)...)
        for cell in cellpoints]
    xboxdict = Lar.coordintervals(1,bboxes)
    yboxdict = Lar.coordintervals(2,bboxes)
    # xs,ys are IntervalTree type
    xs = IntervalTrees.IntervalMap{Float64, Array}()
    for (key,boxset) in xboxdict
        xs[tuple(key...)] = boxset
    end
    ys = IntervalTrees.IntervalMap{Float64, Array}()
    for (key,boxset) in yboxdict
        ys[tuple(key...)] = boxset
    end
    xcovers = Lar.boxcovering(bboxes, 1, xs)
    ycovers = Lar.boxcovering(bboxes, 2, ys)
    covers = [intersect(pair...)
        for pair in zip(xcovers,ycovers)]

    if dim == 3
        zboxdict = Lar.coordintervals(3,bboxes)
        zs = IntervalTrees.IntervalMap{Float64, Array}()
        for (key,boxset) in zboxdict
            zs[tuple(key...)] = boxset
        end
        zcovers = Lar.boxcovering(bboxes, 3, ys)
        covers = [intersect(pair...) for pair in zip(zco
    end
    # remove each cell from its cover
    for k=1:length(covers)
        covers[k] = setdiff(covers[k],[k])
    end
    return covers
end
```

Parametric intersection of lines w the closest ones

lineparams = linefragments(V,EV,Sigma)

```
julia> Sigma = Lar.spaceindex(model)
20-element Array{Array{Int64,1},1}:
 [19]
 [19]
 [7]
 []
 []
 [19, 13]
 [3]
 [17]
 [12]
 []
 []
 [9]
 [19, 6]
 []
 []
 []
 [8]
 []
 [13, 6, 2, 1]
 []
```

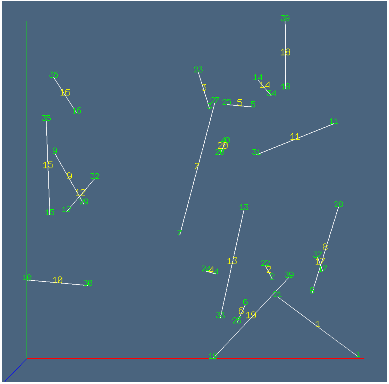


Figure 7: Spatial index computation

Accelerated line intersection

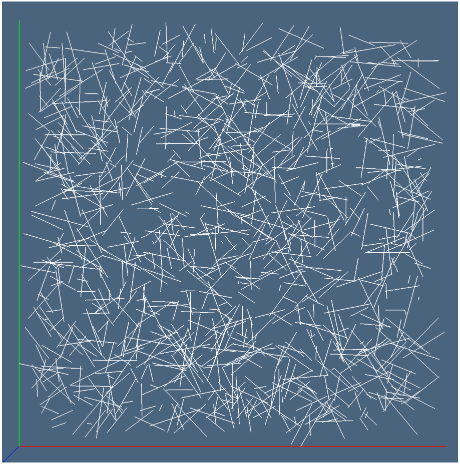


Figure 8: Spatial index computation

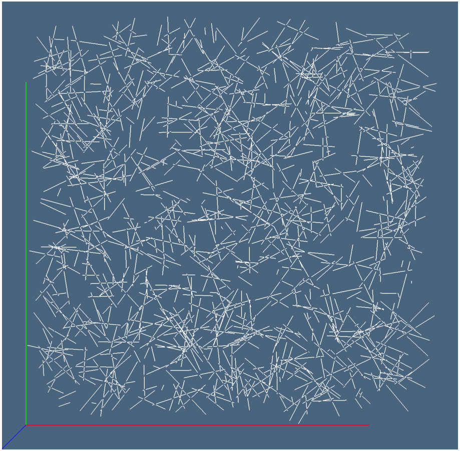


Figure 9: Line segment intersection

Planar graph by congruence

From independently generated line segments to their graph

Homology relation

Two $(d-1)$-spaces (curves, surfaces, etc.) embedded in $\mathbb{E}^d$ are topologically *homologous* when their boundaries can be glued, enclosing a portion of the ambient space, and subdivide $\mathbb{E}^d$ in two parts, inner and outer.

From independently generated line segments to their graph

Homology relation

Two $(d-1)$-spaces (curves, surfaces, etc.) embedded in $\mathbb{E}^d$ are topologically *homologous* when their boundaries can be glued, enclosing a portion of the ambient space, and subdivide $\mathbb{E}^d$ in two parts, inner and outer.

Congruence relation

Two geometric figures are geometrically *congruent* iff one can be transformed into the other by an isometry.

Congruences are equivalence relations

Congruences R_p between p-cells of geometric complexes are equivalence relations, so we may compute the chain complex of quotient chain spaces:

$$C_2(U_2/R_2) \xrightarrow{\partial_2} C_1(U_1/R_1) \xrightarrow{\partial_1} C_0(U_0/R_0),$$

- over which subsequently build the yet unknown basis of C_3

Congruences are equivalence relations

Congruences R_p between p-cells of geometric complexes are equivalence relations, so we may compute the chain complex of quotient chain spaces:

$$C_2(U_2/R_2) \xrightarrow{\partial_2} C_1(U_1/R_1) \xrightarrow{\partial_1} C_0(U_0/R_0),$$

- over which subsequently build the yet unknown basis of C_3
- $^*C_p(U_p/R_p)$ stands for the chain space generated by $X_p = U_p/R_p$.

Congruences are equivalence relations

Congruences R_p between p-cells of geometric complexes are equivalence relations, so we may compute the chain complex of quotient chain spaces:

$$C_2(U_2/R_2) \xrightarrow{\partial_2} C_1(U_1/R_1) \xrightarrow{\partial_1} C_0(U_0/R_0),$$

- over which subsequently build the yet unknown basis of C_3
- *$C_p(U_p/R_p)$ stands for the chain space generated by $X_p = U_p/R_p$.
- in this stage we compute, for each $\sigma \in \mathcal{S}_2$, the quotient sets and the maps ∂_p in-between, for $p = 0, 1, 2$.

PARAMETRIC !! generation of random test data

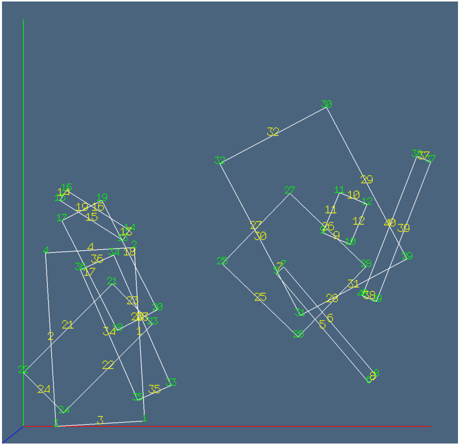


Figure 10: Some examples

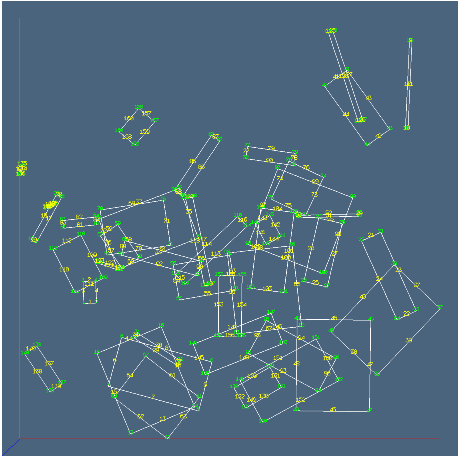


Figure 11: Some examples

Random 2D cuboids in `examples/randomshapes.jl`

using LinearAlgebraicRepresentation Lar = LinearAlgebraicRepresentation
using Plasm

```
function cuboids(n,scale=1.)
    assembly = []
    for k=1:n
        corner = rand(Float64, 2)
        sizes = rand(Float64, 2)
        V,(_,EV,_) = Lar.cuboid(corner,true,corner+sizes)
        center = (corner + corner+sizes)/2
        angle = rand(Float64)*2*pi
        obj = Lar.Struct([ Lar.t(center...), Lar.r(angle),
                Lar.s(scale,scale), Lar.t(-center...), (V,EV)
        push!(assembly, obj)
    end
    Lar.struct2lar(Lar.Struct(assembly))
end
```

Generation of parametric test data

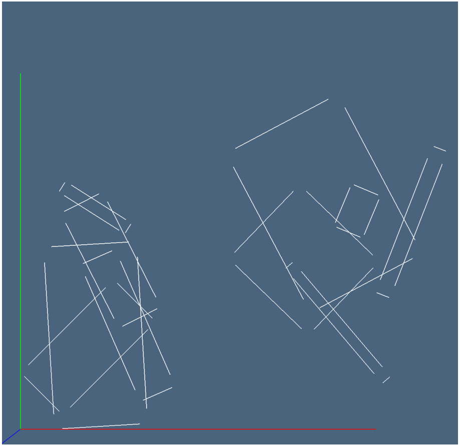


Figure 12: Some example

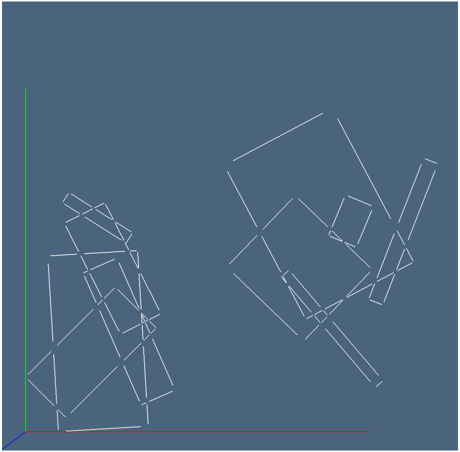
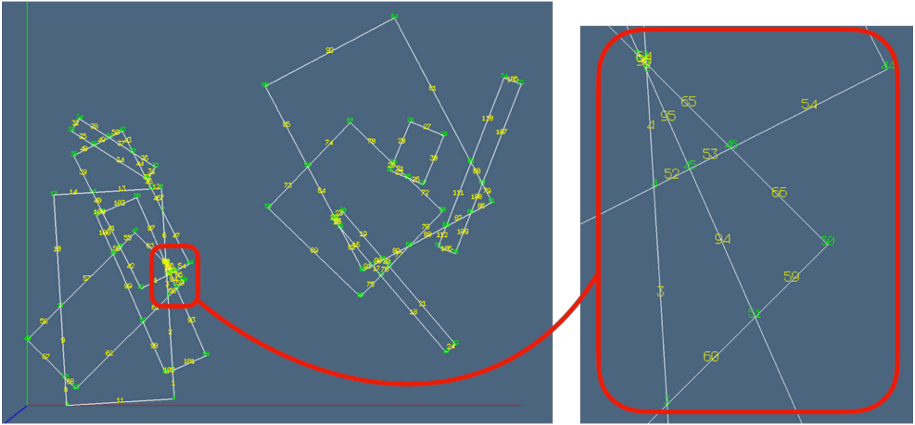


Figure 13: Some example

Visual correctness test

A formal (statistical) proof will be only possible using the Euler formula $V - E + F$ after the construction of the 2-complex is completed ...



Merge results via search for local neighbor

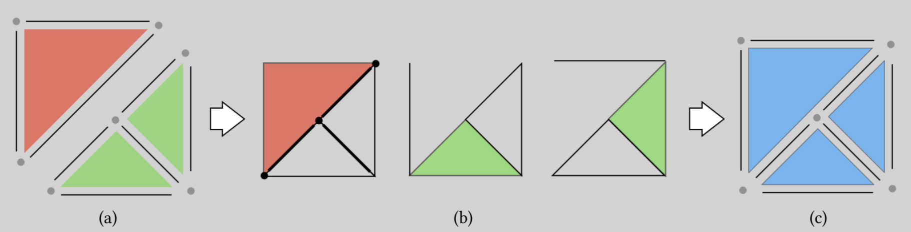


Figure 15: two 2D complexes with incompatible boundaries

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

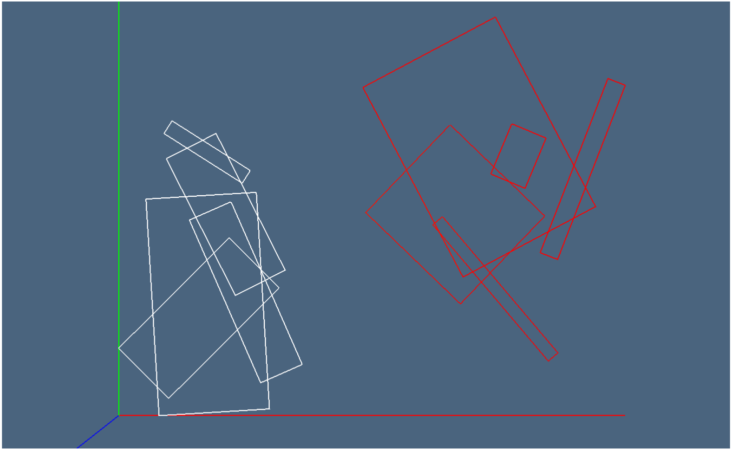


Figure 16: connected components

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

$$Aop^* B = \overline{(A \overset{\circ}{op} B)}$$

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

$$A op^* B = \overline{(A \mathring{op} B)}$$

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

$$A op^* B = \overline{(A \mathring{op} B)}$$

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

bbbbbbb

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

bbbbbbb

bbbbbbb

bbbbbbb