Parallel and Distributed Computing

Alberto Paoluzzi – Lecture 7

Monday 21-03-2022

Arrangements and Boolean algebras 3/3

1. LAR cuboids & simplices
2. LAR maps
3. LAR Integrals
4. LAR Struct
5. LAR splitting 2D
6. LAR TGW 2D
7. LAR congruence
8. LAR TGW 3D
9. LAR generators->bins

Section 1

LAR cuboids & simplices

LAR cuboids & simplices

Generation of multidimensional cuboidal and simplicial complexes, and Cartesian product of complexes

Source file

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/src/largrid.j
https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/src/simplexn

Test file

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/test/largrid.
https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/test/simplex

Docs file

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/docs/src/lar
https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/docs/src/sim

LinearAlgebraicRepresentation.jl/docs/src/largrid.md

https://github.com/cvdlab/LinearAl . . . jl/blob/master/docs/src/largrid.md

https://github.com/cvdlab/LinearAl
. . . jl/blob/master/docs/src/simplexn.md

Example

```
using LinearAlgebraicRepresentation, ViewerGL
Lar = LinearAlgebraicRepresentation; GL = ViewerGL

#   1D complex in 2D generated by Lar and PLaSM operators
circle1D = Lar.circle(.5,2pi)([16])
GL.VIEW([ GL.GLFrame2, GL.GLGrid( circle1D...,GL.COLORS[1],1 ) ]);

#   1D and 0D complex in 1D generated by Lar and PLaSM operators
interval1 = Lar.qn(2)([1])
GL.VIEW([ GL.GLFrame2, GL.GLGrid( interval1...,GL.COLORS[1],1 ) ]);
interval0 = ([0.0 2.0], [[1], [2]])
GL.VIEW([ GL.GLFrame2, GL.GLGrid( interval0...,GL.COLORS[1],1 ) ]);

#   2D complex in 3D generated by Lar and PLaSM operators
cylinder = Lar.larModelProduct([ circle1D, interval1 ])
GL.VIEW([ GL.GLFrame2, GL.GLGrid( cylinder...,GL.COLORS[1],1 ) ]);
disk = Lar.disk(.5,2pi)([16,1])
GL.VIEW([ GL.GLFrame2, GL.GLHull( disk...,GL.COLORS[1],1 ) ]);
tops = Lar.larModelProduct([ disk, interval0 ])
GL.VIEW([ GL.GLFrame2, GL.GLGrid( tops...,GL.COLORS[1],1 ) ]);
brep = Lar.struct2lar(Lar.Struct([cylinder,tops]))
GL.VIEW([ GL.GLFrame2, GL.GLHull( brep...,GL.COLORS[1],1 ) ]);  # KO !!
```

Docs

Documentation on simplicial complexes

ACM-ToG-93.pdf

GP4CAD – ch.4

Input / Output

from a symbolic representation $\Rightarrow$ cellular complex

OR

Lar models $A, B \Rightarrow$ Lar model $A \times B$

Section 2

LAR maps

LAR maps

Generation of parametric primitive geometric shapes, both in 2D and 3D

Source file

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

Test file

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/test/mappe

Docs file

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

LinearAlgebraicRepresentation.jl/docs/src/mapper.md

https://github.com/cvdlab/LinearAl
. . . jl/blob/master/docs/src/mapper.md

Input / Output

from maps $\mathbb{E}^d \to \mathbb{E}^d$ or $\mathbb{E}^d \to \mathbb{E}^{d+1}$

and

cellular complex (usually a 2D or 3D grid)

$\Rightarrow$

polyhedral approximation of a curved (triangulated) complex

API

```
function approxVal(PRECISION)
function approxVal0(value)
function simplifyCells(V,CV)
function circle(radius=1., angle=2*pi)
function circle0(shape=[36])
function helix(radius=1., pitch=1., nturns=2)
function helix0(shape=36*nturns)
function disk(radius=1., angle=2*pi)
function disk0(shape=[36, 2])
function helicoid(R=1., r=0.5, pitch=1., nturns=2)
function helicoid0(shape=[36*nturns, 2])
function ring(r=1., R=2., angle=2*pi)
function ring0(shape=[36, 1])
function cylinder(radius=.5, height=2., angle=2*pi)
function cylinder0(shape=[36, 1])
function sphere(radius=1., angle1=pi, angle2=2*pi, surface=triangled)
function sphere0(shape=[18, 36])
function toroidal(r=1., R=2., angle1=2*pi, angle2=2*pi)
function toroidal0(shape=[24, 36])
function crown(r=1., R=2., angle=2*pi)
function crown0(shape=[12, 36])
function cuboid(maxpoint::Array, full=false,
function ball(radius=1, angle1=pi, angle2=2*pi)
function ball0(shape=[18, 36, 4])
function rod(radius=1., height=3., angle=2*pi)
function rod0(shape=[36, 1])
function hollowCyl(r=1., R=2., height=6., angle=2*pi)
function hollowCyl0(shape=[36, 1, 1])
function hollowBall(r=1., R=1., angle1=pi, angle2=2*pi)
function hollowBall0(shape=[24, 36, 3])
function torus(r=1., R=2., h=.5, angle1=2*pi, angle2=2*pi)
function torus0(shape=[24, 36, 4])
```

Section 3

LAR Integrals

LAR Integrals

Computation of surface and volume integrals of polynomials over cellular complexes

Source file

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

Test file

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/test/integr.j

Docs file

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

LinearAlgebraicRepresentation.jl/docs/src/integr.md

https://github.com/cvdlab/LinearAl . . . jl/blob/master/docs/src/integr.md

Input / Output

from boundary representation

to boundary triangulation

to surface, volume, moments of inertia

API

```
function M(alpha::Int, beta::Int)::Float64
function TT(
    tau::Array{Float64,2},
    alpha::Int, beta::Int, gamma::Int,
    signedInt::Bool=false)
function II(
    P::LAR,
    alpha::Int, beta::Int, gamma::Int,
    signedInt=false)::Float64
function III(P::LAR, alpha::Int, beta::Int, gamma::Int)::Float64
function surface(P::Lar.LAR, signedInt::Bool=false)::Float64
function volume(P::LAR)::Float64
function surfIntegration(larModel)
function firstMoment(P::LAR)::Array{Float64,1}
function secondMoment(P::LAR)::Array{Float64,1}
function inertiaProduct(P::LAR)::Array{Float64,1}
function centroid(P::LAR)::Array{Float64,1}
function inertiaMoment(P::LAR)::Array{Float64,1}
function chainAreas(V::Array{Float64,2},EV::Array{Int64,2})
function chainAreas(V::Array{Float64,2}, EV::Array{Int64,2},...
```

Section 4

LAR Struct

LAR Struct

Hierarchical assemblies of cellular complexes, using affine transformations.
A recursive data structure `Struct` (note the upper-case) is defined.

Source file

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

Test file

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/master/test/struct.j

Docs file

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

LinearAlgebraicRepresentation.jl/docs/src/struct.md

https://github.com/cvdlab/LinearAl . . . jl/blob/master/docs/src/struct.md

Input / Output

From symbolic representation

to data structure

to Lar model or array of models

API

```
function t(args...)
function s(args...)
function r(args...)
function removeDups(CW::Cells)::Cells
function `Struct` to
function Struct()
function Struct(data::Array)
function Struct(data::Array,name)
function Struct(data::Array,name,category)
function name(self::Struct)
function category(self::Struct)
function len(self::Struct)
function getitem(self::Struct,i::Int)
function setitem(self::Struct,i,value)
function pprint(self::Struct)
function set_name(self::Struct,name)
function clone(self::Struct,i=0)
function set_category(self::Struct,category)
function struct2lar(structure) # TODO: extend to true `LARmodels`
function embedTraversal(cloned::Struct,obj::Struct,n::Int,suffix::String)
function embedStruct(n::Int)
function embedStruct0(self::Struct,suffix::String="New")
function box(model)
function apply(affineMatrix, larmodel)
function checkStruct(lst)
function traversal(CTM::Matrix, stack, obj, scene=[])
function evalStruct(self::Struct)
```

Section 5

LAR splitting 2D

LAR splitting 2D

Generation of a 2D arrangement of Euclidean plane (hence of a chain 2-complex) from each face (2-cell) of B-reps of a set of solid 3D shapes. Or from a set of 1D primitives (lines, polygons, circles, etc.)

Source file

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

Test file

https://github.com/cvdlab/Lar.jl/blob/master/test/refactoring.jl

Docs file

https://github.com/cvdlab/Lar.jl/blob/master/docs/src/refactoring.md

Documentation

ACM-TSAS-2020

Input / Output

1. Input: either a Lar structure, to be transformed in a soup of polygons (eiter 2D or 1D), or an array of the same objects, all in the same coordinate system

Input / Output

1. Input: either a Lar structure, to be transformed in a soup of polygons (eiter 2D or 1D), or an array of the same objects, all in the same coordinate system
2. Output: a collection of 2D chain complexes (2D arrangements), embedded in two block-diagonal sparse matrices, to provide the input for marshaling by local congruence algorithms.

Documents to summarize

from ACM-TSAS-2020 and CADJ-2021.pdf

In particular, from ACM-TSAS-2020 the section 2.1 2D arrangements generated by 2-cells (Merge)

and,

from CADJ-2021.pdf: section 2.2. Chains and Arrangements

API

```
function for `pointInPolygonClassification` function.
function crossingTest(new::Int, old::Int, count::T, status::Int)::Number where T <: Real
Function to be parallelized ...
function setTile(box)
function tileCode(point)
function pointInPolygonClassification(V,EV)
function pointInPolygonClassification0(pnt)
function input_collection(data::Array)::Lar.LAR
function boundingbox(vertices::Lar.Points)
function coordintervals(coord,bboxes)
function boxcovering(bboxes, index, tree)
function spaceindex(model::Lar.LAR)::Array{Array{Int,1},1}
function intersection(line1,line2)
function linefragments(V,EV,Sigma)
function fragmentlines(model)
function fraglines(sx::Float64=1.2,sy::Float64=1.2,sz::Float64=1.2)
function fraglines0(model)
function congruence(model)
function decomposition(model::Lar.LAR)
function submanifoldmap(vs)
function Connection
function Bases
function Boundaries
function Containment
function Reduction
function Adjoining
function Assembling
function Output
```

Section 6

LAR TGW 2D

LAR TGW 2D

Topological Gift Wrapping algorithm in 2D, producing a set of chain complexes in 2D.

1. Source file

```
planar_arrangement(V::Points, EV::ChainOp,
[sigma::Chain],
[return_edge_map::Bool],
[multiproc::Bool])
```

https://github.com/cvdlab/Lar.jl/blob/master/src/arrangement/planar_arrang

LAR TGW 2D

Topological Gift Wrapping algorithm in 2D, producing a set of chain complexes in 2D.

1. Source file

```
planar_arrangement(V::Points, EV::ChainOp,
[sigma::Chain],
[return_edge_map::Bool],
[multiproc::Bool])
```

https://github.com/cvdlab/Lar.jl/blob/master/src/arrangement/planar_arrang

1. Test file

LAR TGW 2D

Topological Gift Wrapping algorithm in 2D, producing a set of chain complexes in 2D.

1. Source file

```
planar_arrangement(V::Points, EV::ChainOp,
[sigma::Chain],
[return_edge_map::Bool],
[multiproc::Bool])
```

https://github.com/cvdlab/Lar.jl/blob/master/src/arrangement/planar_arrang

1. Test file
1. Docs file

Documentation

ACM-TSAS-2020

Input / Output

Given any collection of PL cellular polyhedra in this domain, the computation can be summarized by:

1. extracting the 2-skeletons of polyhedra;
2. merging efficiently all their 2-cells, and
3. computing on each 2-cell σ its local chain complex **

$$C_2 \longleftrightarrow C_1 \longleftrightarrow C_0$$

API

```
function frag_edge_channel(in_chan, out_chan, V, EV, bigPI)
function frag_edge(V, EV::Lar.ChainOp, edge_idx::Int, bigPI)
function intersect_edges(V::Lar.Points, edge1::Lar.Cell, edge2::Lar.Cell)
function merge_vertices!(V::Lar.Points, EV::Lar.ChainOp, edge_map, err=1e-4)
function biconnected_components(EV::Lar.ChainOp)
function an_edge(point) # TODO: fix bug
function get_head(edge, tail)
function v_to_vi(v)
function get_external_cycle(V::Lar.Points, EV::Lar.ChainOp, FE::Lar.ChainOp)
function pre_containment_test(bboxes)
function prune_containment_graph(n, V, EVs, shells, graph)
function transitive_reduction!(graph)
function cell_merging(n, containment_graph, V, EVs, boundaries, shells, shell_bboxes)
function bboxes(V::Lar.Points, indexes::Lar.ChainOp)
function decomposition
function componentgraph(V, copEV, bicon_comps)
function cleandecomposition(V, copEV, sigma, edge_map)
function planar_arrangement_1( V, copEV,
function planar_arrangement_2(V, copEV,bicon_comps, edge_map,
function without the `sigma`, `return_edge_map` and `multiproc` arguments
function return also an `edge_map` which maps the edges of the imput to the one of the output. Defaults to `fal
function planar_arrangement(
```

Documentation

ACM-TSAS-2020

Section on Topological gift wrapping (TGW) in 2D

in particular 2.3, 2.4

CADJ-2021.pdf

Section on Chains and Arrangements

in particular 2.2. and A.1. Relevant matters (Algorithm 5)

Section 7

LAR congruence

LAR congruence

Computation of topological congruence between a set of 2D chain complexes. Various possible algorithms. Use of GraphBLAS.jl package.

Source file

https://github.com/cvdlab/LocalCongruence.jl/tree/master/src

Test file

https://github.com/cvdlab/LocalCongruence.jl/tree/master/test

Docs file

https://arxiv.org/abs/2004.00046

Documentation

Algorithm 7 (Chain Complex Congruence (CCC))

vspace{5mm}

CADJ-2021.pdf

API

```
module LocalCongruence
    using DataStructures
    using LinearAlgebraicRepresentation
    using NearestNeighbors
    using SparseArrays
    using GraphBLASInterface
    using SuiteSparseGraphBLAS
    using SparseMM
    Lar = LinearAlgebraicRepresentation

    include("./verticesCongruence.jl")
    include("./cea-AA.jl")
    include("./cea-SM.jl")
    include("./cea-GB.jl")

    export chainCongruenceSM, chainCongruenceAA, chainCongruenceGB
end
```

Input / Output

Given any collection of PL cellular polyhedra in this domain, the computation can be summarized by: (1)~extracting the 2-skeletons of polyhedra; merging efficiently all their 2-cells, and computing on each 2-cell σ its local chain complex

$$C_2 \leftrightarrows C_1 \leftrightarrows C_0$$

3. by identifying in 3D via topological congruence (see [**?**]) the multiple instances of p-cells ($0 \leq p \leq 2$); (4) by finally computing a single chain 2-complex partitioning the whole space $\mathbb{E}^3$, and the matrix $[\partial_2]$ of the boundary operator $\partial_2 : C_2 \to C_1$.

Documentation

DelMonte, G., Onofri, E., Scorzelli, G., Paoluzzi, A., 2020. Local congruence of chain complexes. CoRR abs/2004.00046. URL: https://arxiv.org/abs/2004.00046.

Documents to summarize

ACM-TSAS-2020.pdf

3.2 Quotient sets computation (Congruence algorithm)

and

CADJ-2021.pdf

A.1. Relevant matters: Algorithm 7 (Chain Complex Congruence (CCC))

Section 8

LAR TGW 3D

LAR TGW 3D

Topological Gift Wrapping algorithm in 3D, producing a set of chain complexes in 3D.

```
spatial_arrangement(V::Points,
    copEV::ChainOp, copFE::ChainOp;
    [multiproc::Bool])
```

1. Source file

https://github.com/cvdlab/Lar.jl/blob/master/src/arrangement/spatial_arrang

LAR TGW 3D

Topological Gift Wrapping algorithm in 3D, producing a set of chain complexes in 3D.

```
spatial_arrangement(V::Points,
    copEV::ChainOp, copFE::ChainOp;
    [multiproc::Bool])
```

1. Source file

https://github.com/cvdlab/Lar.jl/blob/master/src/arrangement/spatial_arrang

1. Test file

LAR TGW 3D

Topological Gift Wrapping algorithm in 3D, producing a set of chain complexes in 3D.

```
spatial_arrangement(V::Points,
    copEV::ChainOp, copFE::ChainOp;
    [multiproc::Bool])
```

1. Source file

https://github.com/cvdlab/Lar.jl/blob/master/src/arrangement/spatial_arrang

1. Test file
1. Docs file

Documentation

ACM-TSAS-2020

Input / Output

The TGW algorithm in 3D with input the matrix $[\partial_2]$ produces the matrix $[\partial_3^+]$, whose columns %(but one for each connected component) are a basis of the linear subspace Z_2 of 2-cycles.

The 2-cycles which are boundary of holes are linearly combined, to get the set of generators spanning $B_2 \subset Z_2 \subset C_2$, the subspace of 2-chains that are boundaries of 3-chains (see Figure **??**), in particular of atoms of $\mathbb{E}^3$ partition and of outer space.

API

```
function frag_face_channel(in_chan, out_chan, V, EV, FE, sp_idx)
function frag_face(V, EV, FE, sp_idx, sigma)
function merge_vertices(V::Lar.Points, EV::Lar.ChainOp, FE::Lar.ChainOp, err=1e-4)
function filter_fn(face)
function merge_vertices(V::Lar.Points, EV::Lar.ChainOp, FE::Lar.ChainOp, err=1e-4)
function spatial_arrangement_1(
function removeinnerloops(g, nFE)
function spatial_arrangement_2(
function returns the full arranged complex as a list of vertices V and a chain of borders EV, FE, CF.
function spatial_arrangement(
```

Documents to summarize

ACM-TSAS-2020.pdf

In particular:

1. from CADJ-2021.pdf: 3.3 Computation of ∂_2 and ∂_3 (TGW Algorithm)

Documents to summarize

ACM-TSAS-2020.pdf

In particular:

1. from CADJ-2021.pdf: 3.3 Computation of ∂_2 and ∂_3 (TGW Algorithm)
2. from CADJ-2021.pdf: 2.2, 2.3, and A.1. Relevant matters (Algorithm 5)

Section 9

LAR generators->bins

LAR generators->bins

Characterization of generators of a Boolean algebra as binary strings, depending on the subset of atoms they contain.

Source file

https://github.com/cvdlab/Lar.jl/blob/master/src/bool2d.jl
https://github.com/cvdlab/Lar.jl/blob/master/src/bool3d.jl

Test file

https://github.com/cvdlab/Lar.jl/blob/master/test/bool.jl

Docs file

https://github.com/cvdlab/Lar.jl/blob/master/examples/3d/randomcubes.jl

Documentation

CADJ-2021.pdf

3.2.2. Set-Merbership Classification

Input / Output – 1/3

The B_2 elements are one-to-one with the clopen sets of a partition of $\mathbb{E}^3$ and this one is one-to-one with the atoms of a Boolean solid algebra, produced by our original collection of PL polyhedra, generators of this algebra.

Each generator solid has a structure made by disjoint union of some atoms, and is representable as a binary string, whose length equals the total number of atoms.

Input / Output – 2/3

The validity set (the set of valid representations) of our representation scheme is the set B_2, {i.e.,} the subspace of 2-cycles that are boundary of some coherently oriented 3-chain (i.e. of some PL 3-polyhedra).

The domain of the scheme is $\text{dom}(\delta_2|_{B_2})$, with $\delta_2 : C_2 \to C_3$, capturing our idea of solidity using oriented boundaries.

Input / Output – 3/3

The set of 2^n binary strings with length n (the number of atoms) provides a Boolean algebra of sets.

Here we have operations of union and intersection, as well as difference and complementation.

This algebra is isomorphic to the Boolean Algebra of PL polyhedra provided by our generators, and hence isomorphic to the solid algebra that is called Constructive Solid Geometry (CSG).

API

```
function settestpoints2d(W,EV,FV,f, copEV,copFE) # W by rows
function getinternalpoint2d(W,EV,FV, f, copEV,copFE) # W by rows
function chainbasis2polygons(V,copEV,copFE)
function internalpoints2d(W,copEV,copFE) # W by rows
function testinternalpoint2d(listOfModels)
function testinternalpoint0(testpoint)
function bool2d(assembly)
function boolops(assembly::Lar.Struct, op::Symbol)

function spaceindex(point3d::Array{Float64,1})::Function
function spaceindex0(model::Lar.LAR)::Array{Int,1}
function rayintersection(point3d)
function rayintersection0(V, FV, face::Int)
function planemap(V,copEV,copFE,face)
function planemap0(point)
function settestpoints(V,EV,FV,Fs, copEV,copFE)
function testinternalpoint(V,EV,FV)
function testinternalpoint0(testpoint)
function getinternalpoint(V,EV,FV,Fs, copEV,copFE)
function chainbasis2solids(V,copEV,copFE,copCF)
function internalpoints(V,copEV,copFE,copCF)
function bool3d(assembly)
function bool3d(expr, assembly)
```

Documents to summarize

3. Solid Algebras and Boundary Chains

In particular, from CADJ-2021.pdf: 3.2.2, 3.2.3, 3.2.4, and 3.3.1