Computational topology: Lecture 11

Alberto Paoluzzi

April 6, 2019

1. Lab work

2. Home work

Lab work

Fixing bug in function: it loops !!

continue from previous lecture

```
function buildFV(copEV::ChainOp, face::Cell)
    startv = -1
    nextv = 0
    edge = 0
    vs = Array{Int64, 1}()
    while startv != nextv
        if startv < 0
            edge = face.nzind[1]
            startv = copEV[edge,:].nzind[face[edge] < 0 ? 2 : 1]
            push!(vs, startv)
        else
            edge = setdiff(intersect(face.nzind, copEV[:, nextv].nzind),
                edge)[1]
        end
        nextv = copEV[edge,:].nzind[face[edge] < 0 ? 1 : 2]
        push!(vs, nextv)
    end
    return vs[1:end-1]
end
```

Bug fixed

just run:

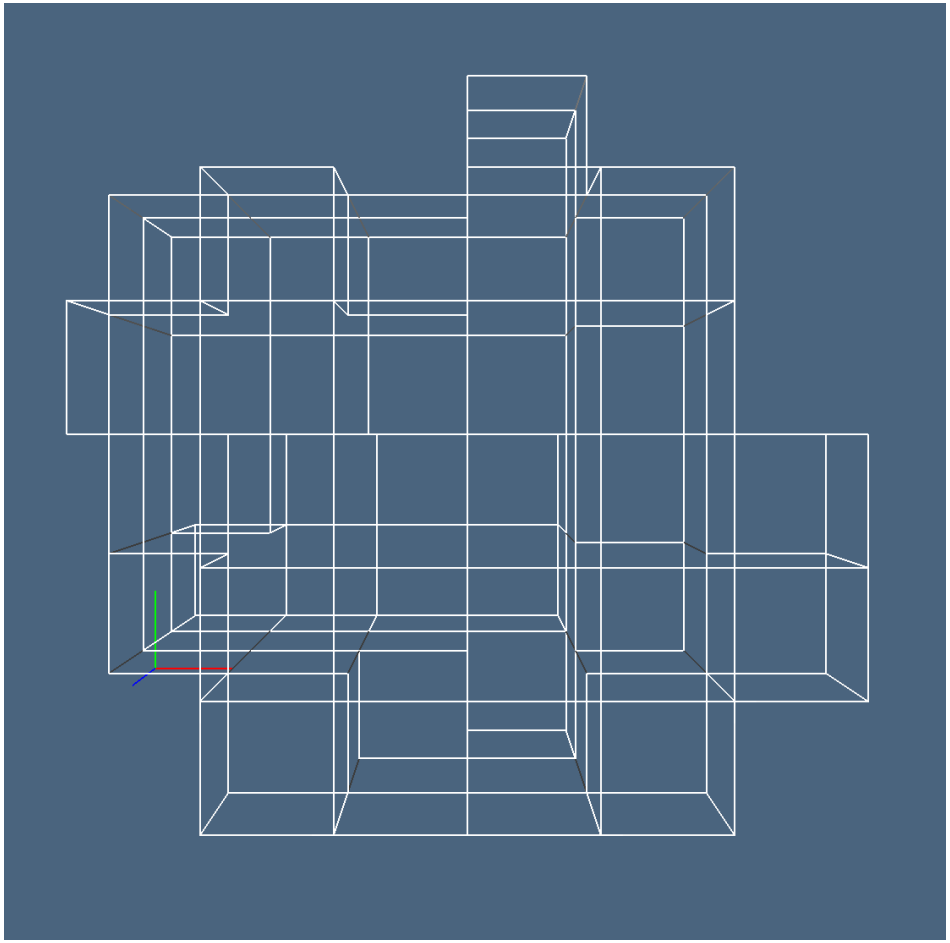
```
$ cd ~/Documents/dev/LinearAlgebraicRepresentation.jl/
$ julia11 examples/lario2obj.jl
```

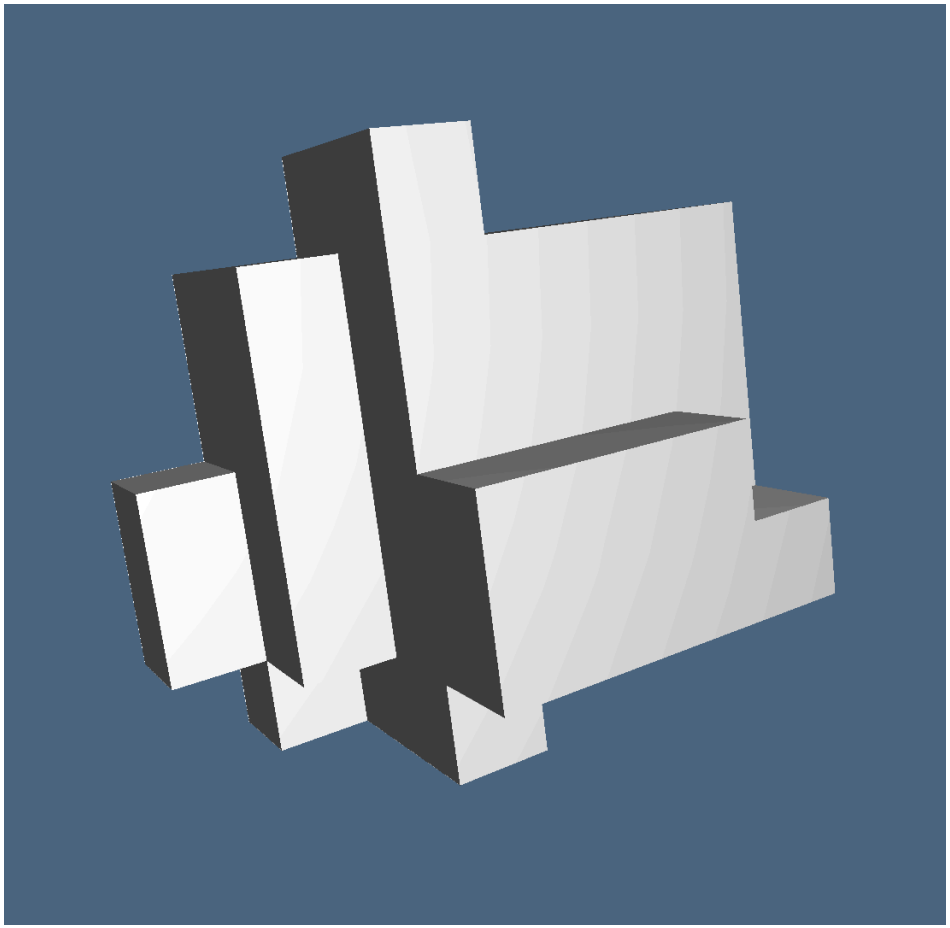
Sparse matrix internals 4/4

evaluate step-by-step in the terminal

```
function vcycle( copEV::Lar.ChainOp, copFE::Lar.ChainOp, f::Int64 )
    edges,signs = findnz(copFE[f,:])
    vpairs = [s>0 ? findnz(copEV[e,:])[1] : reverse(findnz(copEV[e,:])[1])
                for (e,s) in zip(edges,signs)]
    vdict = Dict((v1,v2) for (v1,v2) in  vpairs)

    v0 = collect(vdict)[1][1]
    chain_0 = Int64[v0]
    v = vdict[v0]
    while v !== v0
        push!(chain_0,v)
        v = vdict[v]
    end
    return chain_0
end
```

Visualization from `examples/lario2obj.obj`





Look at sources . . .

and finally go to look at the sources ...

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/julia-1.0/src/utilities.jl

Exporting to file

Write the text file as a single string

```
open("testfile.obj","w") do f
    write(f,Lar.lar2obj(V::Lar.Points, Lar.cc::ChainComplex))
end
```

For reading/writing text files in Julia, see:

Introducing Julia/Working with text files
https://juliabyexample.helpmanual.io

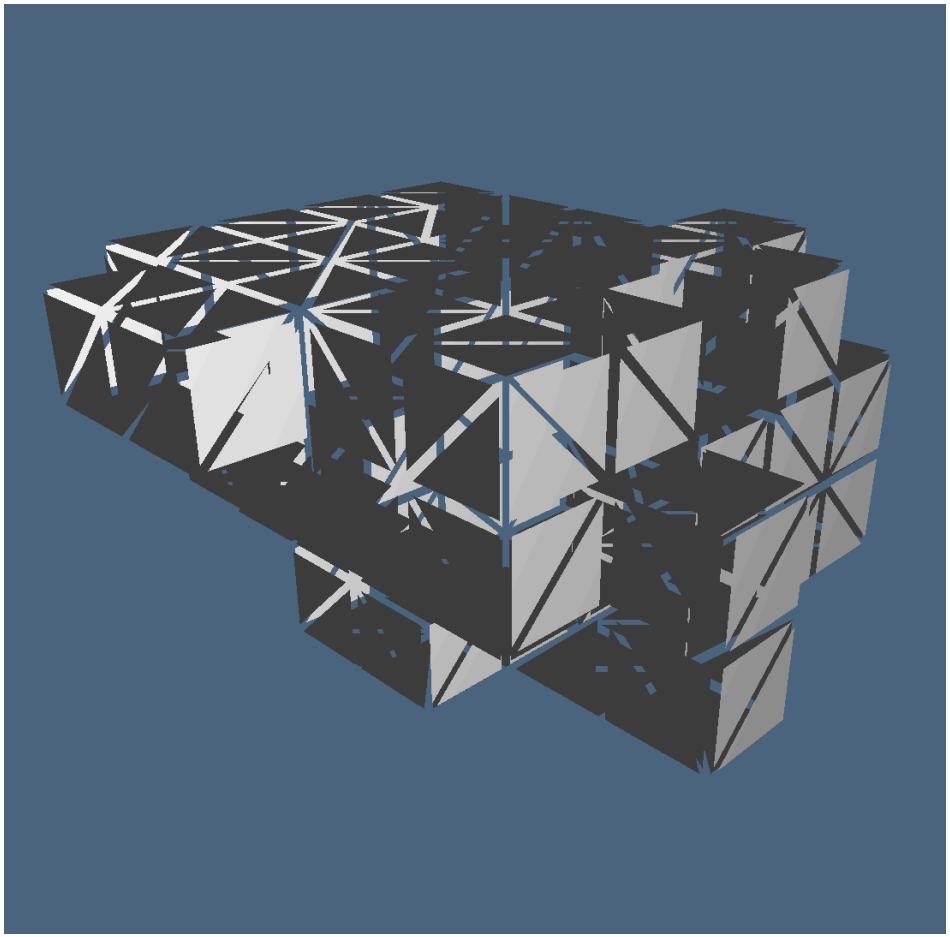
Read/write the text file as a single string

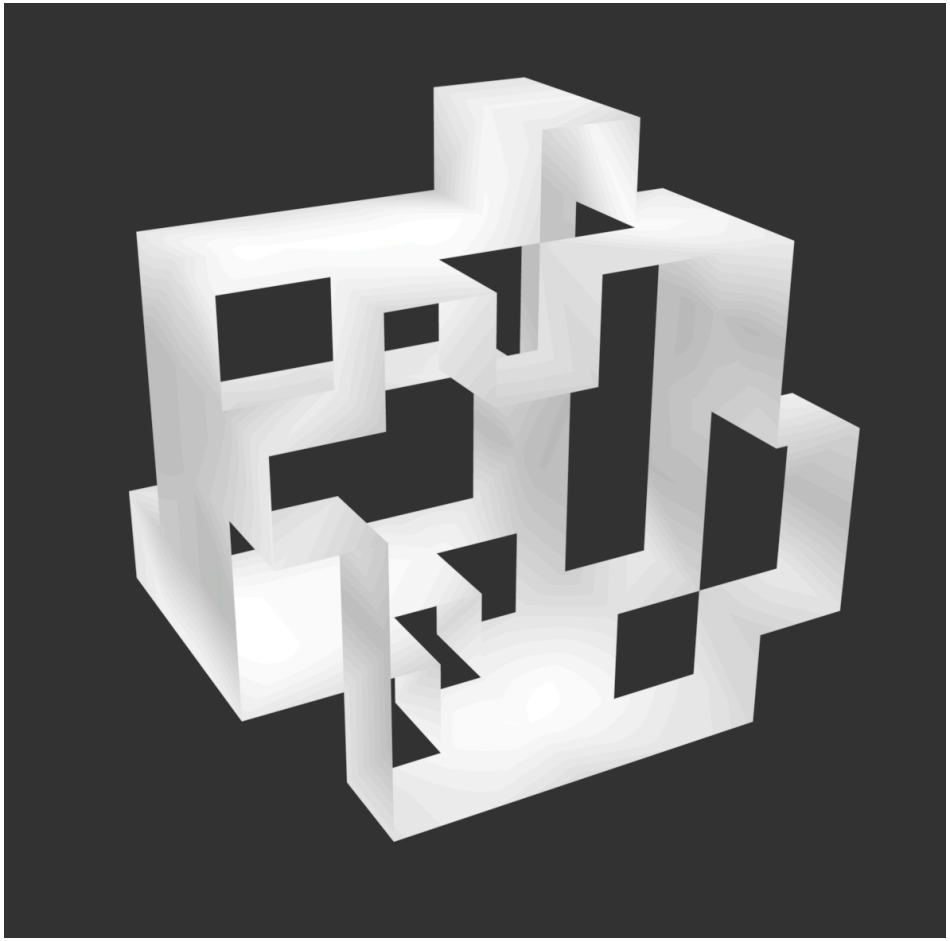
```
s = open("testfile.obj") do file
    read(file, String)
end;

println(s)
```

Visualization from python

```
>>> from pyplasm import *
>>> batches=[]
>>> filename = "testfile.obj"
>>> batches+=Batch.openObj(filename)
>>> octree=Octree(batches)
>>> glcanvas=GLCanvas()
>>> glcanvas.setOctree(octree)
>>> glcanvas.runLoop()

Building octree from 1 batches....
Scene number of nodes of the octree 1
Scene max depth                     0
Scene number of batches             1
...done in 0 msec
```

Visualization of `testfile.obj` from python





Home work

Refactoring job

https://github.com/cvdlab/LinearAlgebraicRepresentation.jl/blob/julia-1.0/test/test_planar_arrangement.jl

Execute and test each function

Save test data and write a new file `test/test_planar_arrangement.jl` using Julia's macro `@testing`

Great Julia Readings . . .

Not only for technical computing: changing the narrative around the usecase for Julia

read also the answer from StefanKarpinski