Computational topology: Lecture 10

Alberto Paoluzzi

April 4, 2019

TGW pseudocode

TGW pseudocode

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

```
/* Pre-condition: d equals the space 𝔼^d dimension, such that (d − 1)-cells are shared by two d-cells */
/*                                                                                                    */
Input: [∂_{d−1}]  # Compressed Sparse Column (CSC) signed matrix (a_ij), where a_ij ∈ {−1, 0, 1}
Output: [∂_d^+]  # CSC signed matrix of cycles
[∂_d^+] = [] ; m, n = [∂_{d−1}].shape ; marks = Zeros(n)   # initializations
while Sum(marks) < 2n do
    σ = Choose(marks)        # select the (d − 1)-cell seed of the column extraction
    if marks[σ] == 0 then [c_{d−1}] = [σ]
    else if marks[σ] == 1 then [c_{d−1}] = [−σ]
    [c_{d−2}] = [∂_{d−1}][c_{d−1}]        # compute boundary c_{d−2} of seed cell
    while [c_{d−2}] ≠ [] do        # loop until boundary becomes empty
        corolla = []
        for τ ∈ c_{d−2} do        # for each “hinge” τ cell
            [b_{d−1}] = [τ]^t[∂_{d−1}]        # compute the τ coboundary
            pivot = {|b_{d−1}|} ∩ {|c_{d−1}|}        # compute the τ support
            if τ > 0 then adj = Next(pivot, Ord(b_{d−1}))        # compute the new adj cell
            else if τ < 0 then adj = Prev(pivot, Ord(b_{d−1}))
            if ∂_{d−1}[τ, adj] ≠ ∂_{d−1}[τ, 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}] = [∂_{d−1}][c_{d−1}]        # compute again the boundary of c_{d−1}
    end
    for σ ∈ c_{d−1} do marks[σ] += 1        # update the counters of used cells
    [∂_d^+] += [c_{d−1}]        # append a new column to [∂_d^+]
end
return [∂_d^+]
```

Figure 1: Example step (d)

Exporting LAR models to Obj format

Wavefront .obj file

From [Wikipedia])[https://en.wikipedia.org/wiki/Wavefront_.obj_file]

OBJ geometry format

1. Filename extension .obj

Wavefront .obj file

From [Wikipedia])[https://en.wikipedia.org/wiki/Wavefront_.obj_file]

OBJ geometry format

1. Filename extension .obj
2. Internet media type text/plain

Wavefront .obj file

From [Wikipedia])[https://en.wikipedia.org/wiki/Wavefront_.obj_file]

OBJ geometry format

1. Filename extension .obj
2. Internet media type text/plain
3. Developed by Wavefront Technologies

Wavefront .obj file

From [Wikipedia])[https://en.wikipedia.org/wiki/Wavefront_.obj_file]

OBJ geometry format

1. Filename extension .obj
2. Internet media type text/plain
3. Developed by Wavefront Technologies
4. Type of format 3D model format

Wavefront .obj file

From [Wikipedia])[https://en.wikipedia.org/wiki/Wavefront_.obj_file]

OBJ geometry format

1. Filename extension .obj
2. Internet media type text/plain
3. Developed by Wavefront Technologies
4. Type of format 3D model format
5. OBJ (or .OBJ) is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package.

Wavefront .obj file

From [Wikipedia])[https://en.wikipedia.org/wiki/Wavefront_.obj_file]

OBJ geometry format

1. Filename extension .obj
2. Internet media type text/plain
3. Developed by Wavefront Technologies
4. Type of format 3D model format
5. OBJ (or .OBJ) is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package.
6. The file format is open and has been adopted by other 3D graphics application vendors.

Wavefront .obj file

The OBJ file format is a simple data-format that represents 3D geometry alone:

- the position of each vertex,

and the

Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary

OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line

Wavefront .obj file

The OBJ file format is a simple data-format that represents 3D geometry alone:

- the position of each vertex,
- the UV position of each texture coordinate vertex,

and the

Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary

OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line

Wavefront .obj file

The OBJ file format is a simple data-format that represents 3D geometry alone:

- the position of each vertex,
- the UV position of each texture coordinate vertex,
- vertex normals,

and the

Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary

OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line

Wavefront .obj file

The OBJ file format is a simple data-format that represents 3D geometry alone:

- the position of each vertex,
- the UV position of each texture coordinate vertex,
- vertex normals,

and the

- faces that make each polygon defined as a list of vertices, and

Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary

OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line

Wavefront .obj file

The OBJ file format is a simple data-format that represents 3D geometry alone:

- the position of each vertex,
- the UV position of each texture coordinate vertex,
- vertex normals,

and the

- faces that make each polygon defined as a list of vertices, and
- texture vertices

Vertices are stored in a counter-clockwise order by default, making explicit declaration of face normals unnecessary

OBJ coordinates have no units, but OBJ files can contain scale information in a human readable comment line

Lab work

Fixing bug in function: loops !!

```
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
```

Fixing bug in function: test data

```
julia> face = sparsevec([1,2,5,12],[1,1,1,1],size(FV,1))

julia> face
264-element SparseVector{Int8,Int64} with 4 stored entries:
  [1  ]  =  1
  [2  ]  =  1
  [5  ]  =  1
  [12 ]  =  1

julia> EV[1]
2-element Array{Int64,1}:
 66
 67

julia> EV[2]
2-element Array{Int64,1}:
 66
 73

julia> EV[5]
2-element Array{Int64,1}:
 67
 74

julia> EV[12]
2-element Array{Int64,1}:
 73
 74
```

Sparse matrix internals 1/4

```
julia> vertpairs = [EV[e] for e in edges]
4-element Array{Array{Int64,1},1}:
 [1, 2]
 [3, 4]
 [5, 6]
 [10, 12]

julia> verts = Set(cat(vertpairs))
Set(Any[12, 4, 10, 2, 3, 5, 6, 1])

julia> vertedges = [(v1,e) for (e,(v1,v2)) in zip(edges,vertpairs)]
4-element Array{Tuple{Int64,Int64},1}:
 (1, 1)
 (3, 2)
 (5, 5)
 (10, 12)

julia> edgeverts = [(e,v2) for (e,(v1,v2)) in zip(edges,vertpairs)]
4-element Array{Tuple{Int64,Int64},1}:
 (1, 2)
 (2, 4)
 (5, 6)
 (12, 12)
```

Sparse matrix internals 2/4

```
julia> c_1 = sparse([],[],Int8[],1,size(V,2))
1×448 SparseMatrixCSC{Int8,Int64} with 0 stored entries

julia> c_1 * VE
1×264 SparseMatrixCSC{Int8,Int64} with 0 stored entries

julia> c_0 = c_1 * VE
1×264 SparseMatrixCSC{Int8,Int64} with 0 stored entries

julia> c_0
1×264 SparseMatrixCSC{Int8,Int64} with 0 stored entries

julia> copFE = Lar.coboundary_1( V, kFV::Lar.ChainOp, kEV::Lar.ChainOp)
132×264 SparseMatrixCSC{Int8,Int64} with 528 stored entries:

julia> edges,signs = findnz(copFE[1,:])
([1, 2, 5, 12], Int8[1, -1, 1, -1])

julia> [EV[e] for e in edges]
4-element Array{Array{Int64,1},1}:
 [66, 67]
 [66, 73]
 [67, 74]
 [73, 74]
```

Sparse matrix internals 3/4

```
julia> [EV[e] for e in edges]
4-element Array{Array{Int64,1},1}:
 [66, 67]
 [66, 73]
 [67, 74]
 [73, 74]

julia> face = map(prod, zip(edges,signs))
4-element Array{Int64,1}:
   1
  -2
   5
 -12

julia> vpairs = [e>0 ? EV[e] : reverse(EV[-e]) for e in face]
4-element Array{Array{Int64,1},1}:
 [66, 67]
 [73, 66]
 [67, 74]
 [74, 73]

julia> vdict = Dict((v1,v2) for (v1,v2) in  vpairs)
Dict{Int64,Int64} with 4 entries:
  66 => 67
  67 => 74
  74 => 73
  73 => 66
```

```
julia> collect(vdict)
4-element Array{Pair{Int64,Int64},1}:
 66 => 67
 67 => 74
 74 => 73
 73 => 66

julia> collect(vdict)[1]
66 => 67

julia> v = collect(vdict)[1][1]
66
```

Sparse matrix internals 4/4

```
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
```

Look at sources . . .

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