Geometric model extraction from 3D medical data with Visus+LAR architecture [*]

Giulia Clementi, CVD Lab, Roma Tre University

March 22, 2016

Abstract

This document describes the Visuspy+LAR architecture.

Contents

[*] This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework. March 22, 2016

1 Visuspy+LAR streaming architecture

Data preparation

- the visus query on the *idx* file extracts slabs of 3D venous/neural data at finest resolution as a three-dimensional row-major matrix;
- application of a threshold on the data;
- denoising via median filter, if needed.

Model generation

- at a certain time only three adiacent slabs of the dataset are loaded into memory. They are managed by different processes.
- every slab is divided into bricks;
- boundary extraction within every brick through an operation of SpMV multiplication between the CSR representation of $[\partial_3]$ and the CSC representation of the 3-chain;
- double faces removal of the median slab, where the upper slab has already been cleaned and smoothed;
- laplacian smoothing on the median slab. The slab that lays downwards has been cleaned only in its upper part;
- use of obj format to save the extracted model.

Visualization layer

- at a certain moment the upper slab is ready and can be loaded and visualized on the fly;
- lar or pyplasm visualization.

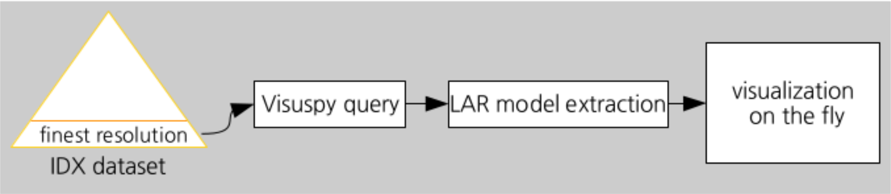

Figure 1: *Visuspy+LAR software architecture*

2 Prerequisites installation

Recompile Visus and activate the python flag.

⟨ Prerequisites 3a ⟩ ≡

```
        cd path/to/nvisusio/build/linux
        ccmake ../../
        # activate the option VISUS_BUILD_SWIG_PYTHON
        # configure
        # exit
        # generate
        # quit
        make -j 8
        make install
     ◇
```

Macro never referenced.

3 Introduction to visuspy query

3.1 Prepare the dataset with Visus Convert

Use Visus Convert to create the *idx* dataset for the next examples.

⟨ Create IDX 3b ⟩ ≡

```
     cd path/to/nvisusio
     CONVERT="build/linux/visusconvert"
     RESOURCES="resources"
     $CONVERT --import $RESOURCES/tutorials/cat_rgb.tga  --create temp/cat_rgb.idx
     ◇
```

Macro never referenced.

3.2 Basic 2D query

This first example creates a 2D box, performs the query and visualizes the image of the cat through *matplotlib*. Select the *idx* dataset file, the field and the size of the box. The *set* method takes as parameters the dimension and the coordinate: 0 is x, 1 is y, 2 is z. The box is 256 on x and 256 on y. The query output is a row-major array.

⟨ Visuspy query 4 ⟩ ≡

```
#cd path/to/nvisusio
#python docs/examples/visuspy/query.py
from visuspy import *
import matplotlib.pyplot as plt

   def query(self):

         #input file & dataset
         filename="temp/cat_rgb.idx"
         dataset=Dataset_loadDataset(filename)

         logic_box=dataset.getLogicBox()
         field=Field("data",DType("uint8[3]"))
         access=dataset.createAccess()

      #box dimension
      box=NdBox(logic_box)
         box.setP1(0,0)
         box.setP1(1,0)
         box.setP2(0,255)
         box.setP2(1,255)

      #perform the query
         query=Query(dataset,ord('r'))
         query.setLogicPosition(Position(box))
         query.setAccess(access)

      query.begin()
         self.assert_(query.execute())
      data=query.getBuffer().get().asNumPyArray()
            print data

      #visualize the image
      plt.imshow(data)
      plt.show()
```

◇

Macro never referenced.

3.3 Manage different resolutions

Decide the resolution of the query output. Visualize the cat at coarser and then full resolution.

⟨ Different resolutions 5 ⟩ ≡

```
from visuspy import *
import matplotlib.pyplot as plt
   def resolution(self):

         #input file & dataset
         filename="temp/cat_rgb.idx"
         dataset=Dataset_loadDataset(filename)
         self.assert_(dataset)

         logic_box=dataset.getLogicBox()
         field=Field("data",DType("uint8[3]"))
         access=dataset.createAccess()

      #box dimension
      box=NdBox(logic_box)
         box.setP1(0,0)
         box.setP1(1,0)
         box.setP2(0,255)
         box.setP2(1,255)

      #get the maximum resolution
      MaxH=dataset.getBitmask().getMaxResolution()
      print "MaxH = "+str(MaxH)

      #perform the query
         query=Query(dataset,ord('r'))
         query.setLogicPosition(Position(box))
         query.setAccess(access)
         query.addEndResolution(MaxH-4)
      query.addEndResolution(MaxH)
      query.setMergeMode(Query.InsertSamples)

      #query 1
      query.begin()
         self.assert_(query.execute())
      data=query.getBuffer().get().asNumPyArray()
            print data

      #visualize the image
```

```
        plt.imshow(data)
        plt.show()

        query.next()

        #query 2
           self.assert_(query.execute())
        data2=query.getBuffer().get().asNumPyArray()
              print data2

        #visualize the image
        plt.imshow(data2)
        plt.show()
◇
```

Macro never referenced.

3.4 3D box scan

This sample extract from *IDX* a 3D box and manages the 3D array as a stack of 2D images. The neural dataset is made of two fields: neurons and vessels. Visualize the first 5 images.

⟨ 3D box 6 ⟩ ≡

```
from visuspy import *
import matplotlib.pyplot as plt
   def 3Dbox(self):
      #input file & dataset
         filename="/home/manuel/Scrivania/microscopy_data/file.idx"
         dataset=Dataset_loadDataset(filename)
         self.assert_(dataset)

         logic_box=dataset.getLogicBox()
         field=Field("vessels",DType("uint8"))
         access=dataset.createAccess()

      #box dimension
      box=NdBox(logic_box)
         box.setP1(0,500)
         box.setP1(1,500)
      box.setP1(2,500)
         box.setP2(0,550)
         box.setP2(1,550)
      box.setP2(2,550)

      #get the maximum resolution
```

```
MaxH=dataset.getBitmask().getMaxResolution()
print "MaxH = "
print MaxH

#perform the query
   query=Query(dataset,ord('r'))
   query.setLogicPosition(Position(box))
   query.setAccess(access)
query.addEndResolution(MaxH)
query.begin()
   self.assert_(query.execute())
data3d=query.getBuffer().get().asNumPyArray()

for Z in range(0,5):
   image=data3d[Z]
   plt.imshow(image)
   plt.show()
```

◇

Macro never referenced.

4 Multithreading example

This first iteration is characterized by few main points, that help performance improvent from the first *larVolumeToObj*:

- The entire dataset must not be loaded into memory. The *query* tool allows the loading of slices of the volume;
- We don't want to write the entire dataset on disk in *pklz* format. Data load and 2-chain extraction must have been done at the same time without further writings;
- The slices will be processed in parallel.

In this piece of code we reach this aims through multithreading. Each thread performs the query to obtain a slice of the entire volume and computes the 2-chain of 2-quads for every brick in the slice. The remaining part of the application stays the same, it will be taken into analysis in the next sections.

⟨ Multithreading 7 ⟩ ≡

```
from visuspy import *
import larVolumeToObj
import threading

   def example(self):
```

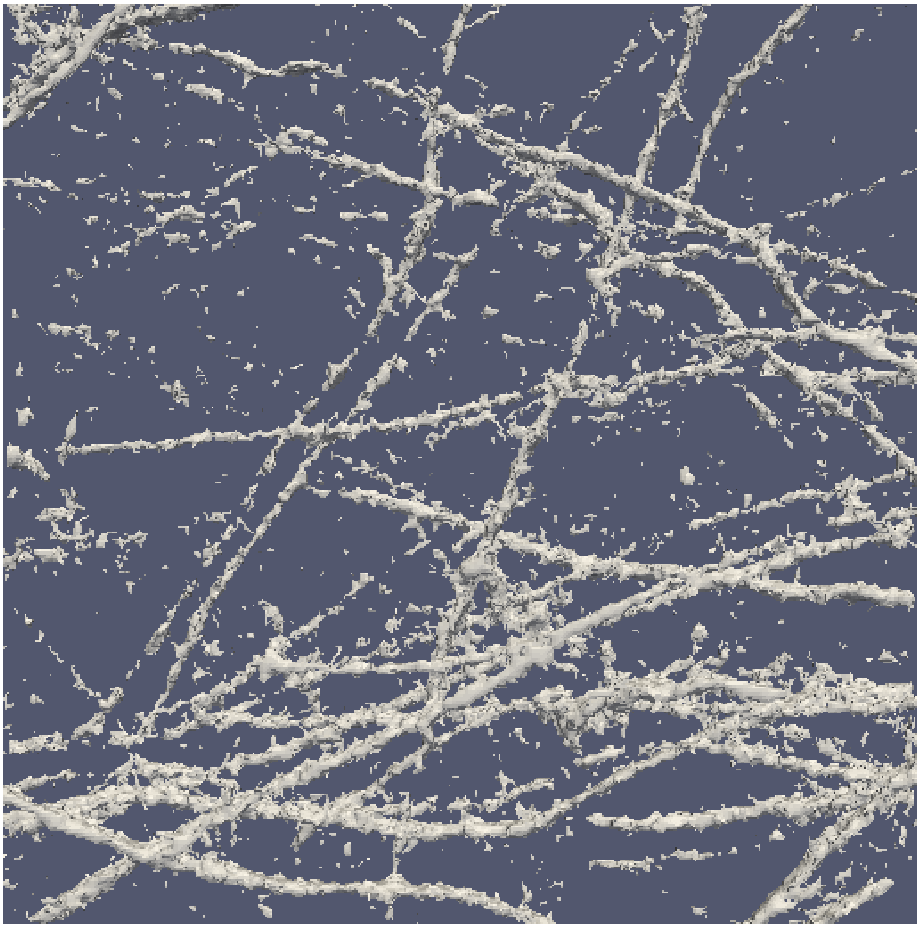

Figure 2: *Model at finest resolution of the venous system of the brain.*

```
#visuspy initialization
app=Application()

#definition of the paths
outputdir=’output’
borderdir=’output/border’
stldir = os.path.join(outputdir, ’stl’)
bindir = os.path.join(outputdir, ’compbin’)
binfile = os.path.join(bindir, ’model-2.bin’)
stlfile = os.path.join(stldir, ’model-2.obj’)

#delete the old output directory
if os.path.exists(outputdir):
      shutil.rmtree(outputdir)

#create the new output directories
self.mkdir_p(stldir)
self.mkdir_p(bindir)
self.mkdir_p(borderdir)

#generate the border matrix
bordo3path = gbmatrix.getOrientedBordo3Path(5, 5, 5,borderdir)

#create and start the threads
#every thread creates the poupout binary
#files with the 2-chain of 2-quads for every brick
threads = []
for n in xrange(5):
      t = myThread(n)
      t.start()
      threads.append(t)

# join all threads
for t in threads:
   t.join()

#concatenate all binary files in model-2.bin
larVolumeToObj.computation.pklzToSmoothObj.concatenate_files(
      "output/*.bin",
   binfile)

#convert model-2.bin in model-2.obj
   sq.make_obj(
   5, 5, 5,
   binfile,
      stldir)
      outputfile='out'
      obj_input = 'stl/model-2.obj'
         larVolumeToObj.computation.pklzToSmoothObj.concatenate_files(
      stldir + '/output-*-*.stl', stlfile)

      #load the model in model-2.obj and print information
      V, F = fileio.readFile(os.path.join(outputdir, obj_input))
      print "Before"
         print "Number of vertexes: %i    Number of faces %i" % (len(V), len(F))

      #make double faces removal and smoothing
         V, F = pklzToSmoothObj.makeCleaningAndSmoothing(
         V, F,
         os.path.join('output/stl', outputfile))

      #print information
         print "After"
         print "Number of vertexes: %i    Number of faces %i" % (len(V), len(F))

      # fill empty vertexes
      V = [v if len(v) == 3 else [0, 0, 0] for v in V]

      # scaling: make ten times bigger
      Vint = (numpy.asarray(V) * 10).astype(numpy.int).tolist()

      #write after scaling
            fileio.writeFile(
               os.path.join(outputdir, outputfile + "_sm_i.obj"),
               Vint, F,
               ignore_empty_vertex_warning=True)

      #triangulate after scaling and save
         Ftr = pklzToSmoothObj.save_triangulated(V, Vint, F, outputdir, outputfile)

      #pyplasm visualization
         larVolumeToObjParallelo.computation.visualization.visualize(V, F, explode=False)


class myThread (threading.Thread):

    def __init__(self, threadID):
        threading.Thread.__init__(self)
        self.threadID = threadID

    def run(self):
   #visus query
```

```
     filename="/home/manuel/Scrivania/microscopy_data/file.idx"
dataset=Dataset_loadDataset(filename)
logic_box=dataset.getLogicBox()
field=Field("vessels",DType("uint8"))
access=dataset.createAccess()

#parametric box
box=NdBox(logic_box)
box.setP1(0,500)
box.setP1(1,500)
box.setP1(2,500+10*self.threadID)
box.setP2(0,600)
box.setP2(1,600)
box.setP2(2,500+10*self.threadID+10)

#get the maximum resolution
MaxH=dataset.getBitmask().getMaxResolution()

#perform the query
query=Query(dataset,ord(’r’))
query.setLogicPosition(Position(box))
query.setAccess(access)
query.addEndResolution(MaxH)
query.begin()
query.execute()
data3d=query.getBuffer().get().asNumPyArray()

#parameters label and threshold
label=2
threshold=10

#data structure initialization
metadata = {}  # reader.get_metaData()
metadata[’series_number’] = 0  # reader.series_number
metadata[’datadir’] = "/home/manuel/Scrivania/microscopy_data/"
metadata[’voxelsize_mm’] = [100,100,10]
datap = {}
datap[’data3d’]=data3d
datap[’metadata’]=metadata

#median filter application
NOISE_SHAPE_DETECT=3
for page in xrange(datap[’data3d’].shape[0]):
   datap[’data3d’][page] = ndimage.median_filter(
   datap[’data3d’][page], NOISE_SHAPE_DETECT)
```

```
        #thresholding data
        datap['segmentation'] = (datap['data3d'] > threshold).astype(numpy.uint8) * label

        print 'Processing data'

        #2-chain of 2-quad computation
        s2bin.calcchains_main(idslice=self.threadID,
        nx=5, ny=5, nz=5,
        calculateout=True,
        datap=datap,
        BORDER_FILE='output/border/bordo3_5-5-5.json',
        DIR_O='output/compbin',
        coloridx=label,
        label=label,
        )
    ◇
```

Macro defined by 7, 14.
Macro never referenced.

5 Partitioning and cover of a set

The aim of this part is to perform double faces removal and smoothing in parallel for the slabs. We exploit the difference between partition and cover of a set. Following the definitions.

Partition of a set. *A family of sets P is a partition of X if and only if all of the following conditions hold:*

- *P does not contain the empty set;*
- *The union of the sets in P is equal to X. The sets in P are said to cover X;*
- *The intersection of any two distinct sets in P is empty.*

Cover of a set. *A family of sets P is a cover of X if and only if all of the following conditions hold:*

- *P does not contain the empty set;*
- *The union of the sets in P is equal to X.*

5.1 Algorithm idea

- Create the covering for each partition;
- Apply to the covering the standard algorithms;

- Remove what is outside the slab;
- Merge the new part and visualize.

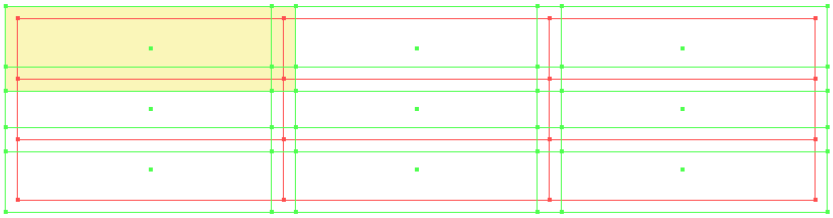

Figure 3: *2D description. red: partitioning on data space (voxels); green: cover, extension of a slab; yellow: one extended slab.*

6 JSON configuration file

Parameters can be set through json configuration file. Here is a very simple sample about decoding data from JSON file.

⟨ JSON decode 13a ⟩ ≡

```
import json

with open('conf.json') as data_file:
    data = json.load(data_file)
crop = data['crop']
print crop[0]
print crop[0][0]
print crop[0][1]
◇
```

Macro never referenced.

⟨ JSON file 13b ⟩ ≡

```
{
    "crop": [[1,800],[1,600],[1,50]]
}
◇
```

Macro never referenced.

7 Nested multithreading and shared data structures

The job is divided between threads. Each thread creates a square, but this square is created by two nested threads. Each nested thread writes a triangle in the same data structure, a list representing a LAR model.

⟨ Multithreading 14 ⟩ ≡

```
import threading
from larlib import *

class CreateTriangle(threading.Thread):
   def __init__(self, idtriangle, idslice , model, lock):
      threading.Thread.__init__(self)
      self.lock = lock
      self.idtriangle = idtriangle
      self.idslice = idslice
      self.model = model
   def run(self):
      self.lock.acquire()
      if self.idtriangle%2==0:
         if [1,0,self.idslice] not in self.model[0]:
            self.model[0].append([1,0,self.idslice])
         if [0,0,self.idslice] not in self.model[0]:
               self.model[0].append([0,0,self.idslice])
         if [0,1,self.idslice] not in self.model[0]:
               self.model[0].append([0,1,self.idslice])
         x = self.model[0].index([1,0,self.idslice])
         y = self.model[0].index([0,0,self.idslice])
         z = self.model[0].index([0,1,self.idslice])
            self.model[1].append([x,y,z])
      else:
         if [1,0,self.idslice] not in self.model[0]:
            self.model[0].append([1,0,self.idslice])
         if [1,1,self.idslice] not in self.model[0]:
               self.model[0].append([1,1,self.idslice])
         if [0,1,self.idslice] not in self.model[0]:
               self.model[0].append([0,1,self.idslice])
         x = self.model[0].index([1,0,self.idslice])
         y = self.model[0].index([0,1,self.idslice])
         z = self.model[0].index([1,1,self.idslice])
            self.model[1].append([x,y,z])
      self.lock.release()

class CreateSlice(threading.Thread):
   def __init__(self, idslice, model, lock):
```

```
            threading.Thread.__init__(self)
            self.lock = lock
            self.idslice = idslice
            self.model = model
         def run(self):
            threads = []
            for idtriangle in xrange(2):
               t = CreateTriangle(idtriangle,self.idslice,self.model,self.lock)
               t.start()
               threads.append(t)
            for t in threads:
               t.join()

      class Esempio:
         def main(self):
            model = [[],[]]
            lock = threading.Lock()

            threads = []
            for idslice in xrange(10):
               t = CreateSlice(idslice,model,lock)
               t.start()
               threads.append(t)
            for t in threads:
               t.join()

            mkpols = MKPOLS(model)
            VIEW(EXPLODE(1.2, 1.2, 1)(mkpols))


      if __name__ == '__main__':
         e = Esempio()
         e.main()
      ◇
```

Macro defined by 7, 14.
Macro never referenced.

8 Final solution

The idea is to divide the computation on each slab in 3 main parts: read through the query, compute the model and write it. The actions on each slab have to ve performed according to the Gant diagram shown in figure.

⟨Final solution 15⟩ ≡

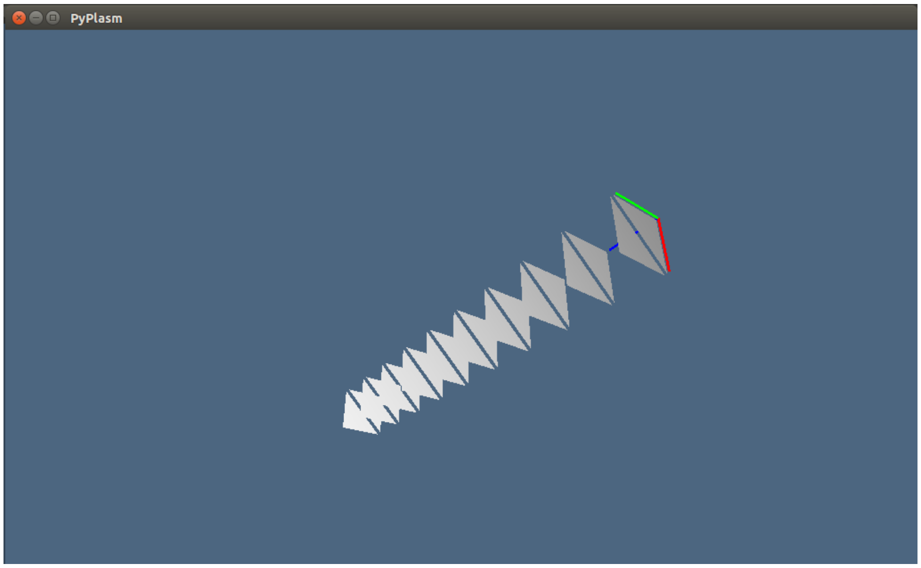

Figure 4: *Output of the nested multithreading sample*

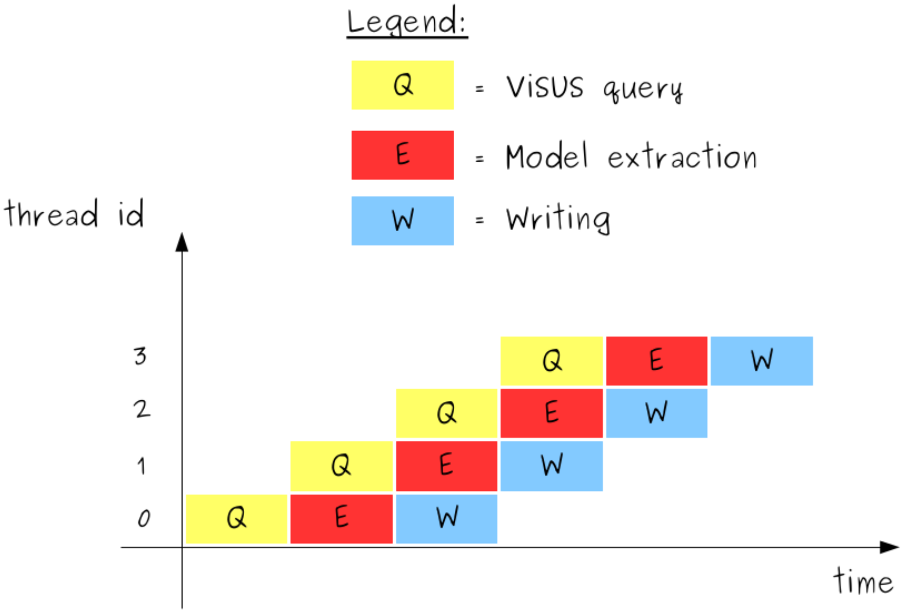

Figure 5: *Gant diagram of the computation.*

◇

Macro never referenced.

9 Generate the boundary operator

The computation is faster with the not-oriented boundary operator, computed through the new lar boundary module.

⟨ Get bordo3 path 17a ⟩ ≡

```
def getBordo3Path(nx, ny, nz, DIR_OUT):
    fileName = DIR_OUT+’/bordo3_’+str(nx)+’-’+str(ny)+’-’+str(nz)+’.json’
    bordo3 = computeBordo3(nx, ny, nz) ⟨Compute bordo3 17b⟩
    writeBordo3(bordo3, fileName) ⟨Write bordo3 19a⟩
    return fileName
```

◇

Macro never referenced.

⟨ Compute bordo3 17b ⟩ ≡

```
def computeBordo3(nx, ny, nz):
    V, [VV, EV, FV, CV] = getBases(nx, ny, nz) ⟨Get basis 17c⟩
    bordo3 = boundary(CV,FV) #LAR
    return bordo3
```

◇

Macro referenced in 17a.

⟨ Get basis 17c ⟩ ≡

```
def getBases(nx, ny, nz):

    def ind(x,y,z): return x + (nx+1) * (y + (ny+1) * (z))

    def the3Dcell(coords):
        x,y,z = coords
        return [ind(x,y,z),ind(x+1,y,z),ind(x,y+1,z),ind(x,y,z+1),ind(x+1,y+1,z),
                ind(x+1,y,z+1),ind(x,y+1,z+1),ind(x+1,y+1,z+1)]

    # Construction of vertex coordinates (nx * ny * nz)
    # ------------------------------------------------------------

    try:
        V = [[x,y,z] for z in xrange(nz+1) for y in xrange(ny+1) for x in xrange(nx+1) ]
    except:
        import ipdb; ipdb.set_trace() #  noqa BREAKPOINT


    log(3, ["V = " + str(V)])

    # Construction of CV relation (nx * ny * nz)
    # ------------------------------------------------------------

    CV = [the3Dcell([x,y,z]) for z in xrange(nz) for y in xrange(ny) for x in xrange(nx)]

    log(3, ["CV = " + str(CV)])

    # Construction of FV relation (nx * ny * nz)
    # ------------------------------------------------------------

    FV = []

    v2coords = invertIndex(nx,ny,nz) ⟨Invert index 18⟩

    for h in xrange(len(V)):
        x,y,z = v2coords(h)
        if (x < nx) and (y < ny): FV.append([h,ind(x+1,y,z),ind(x,y+1,z),ind(x+1,y+1,z)])
        if (x < nx) and (z < nz): FV.append([h,ind(x+1,y,z),ind(x,y,z+1),ind(x+1,y,z+1)])
        if (y < ny) and (z < nz): FV.append([h,ind(x,y+1,z),ind(x,y,z+1),ind(x,y+1,z+1)])

    VV = AA(LIST)(range(len(V)))

    EV = []
    for h in xrange(len(V)):
        x,y,z = v2coords(h)
        if (x < nx): EV.append([h,ind(x+1,y,z)])
        if (y < ny): EV.append([h,ind(x,y+1,z)])
        if (z < nz): EV.append([h,ind(x,y,z+1)])

    return V, (VV, EV, FV, CV)
◇
```

Macro referenced in 17b.

⟨Invert index 18⟩ ≡

```
def invertIndex(nx,ny,nz):
    nx,ny,nz = nx+1,ny+1,nz+1
    def invertIndex0(offset):
        a0, b0 = offset / nx, offset % nx
        a1, b1 = a0 / ny, a0 % ny
```

```
            a2, b2 = a1 / nz, a1 % nz
            return b0,b1,b2
        return invertIndex0
    ◇
```

Macro referenced in 17c, 19b, 26b.

⟨ Write bordo3 19a ⟩ ≡

```
    def writeBordo3(bordo3, inputFile):
        ROWCOUNT = bordo3.shape[0]
        COLCOUNT = bordo3.shape[1]
        ROW = bordo3.indptr.tolist()
        COL = bordo3.indices.tolist()
        DATA = bordo3.data.tolist()

        with open(inputFile, "w") as file:
            json.dump({
                "ROWCOUNT": ROWCOUNT, "COLCOUNT": COLCOUNT,
                "ROW": ROW, "COL": COL, "DATA": DATA}, file,
                separators=(',', ':'))
            file.flush()
    ◇
```

Macro referenced in 17a.

10 Calculate chains

⟨ Calculate chains main 19b ⟩ ≡

```
    def calcchains_main(
        hslice,
        idslice,bin_data,lock,
        nx, ny, nz,
        calculateout,
        datap,
        BORDER_FILE,
        coloridx,
        label
    ):
        def ind(x, y, z):
            return x + (nx+1) * (y + (ny+1) * (z))

        chunksize = nx * ny + nx * nz + ny * nz + 3 * nx * ny * nz
        V = [[x, y, z]
             for z in xrange(nz + 1)
             for y in xrange(ny + 1)
```

```
         for x in xrange(nx + 1)]

    v2coords = invertIndex(nx, ny, nz) ⟨Invert index 18⟩

    # construction of vertex grid
    FV = []
    for h in xrange(len(V)):
        x, y, z = v2coords(h)
        if (x < nx) and (y < ny):
            FV.append([h, ind(x+1, y, z), ind(x, y+1, z), ind(x+1, y+1, z)])
        if (x < nx) and (z < nz):
            FV.append([h, ind(x+1, y, z), ind(x, y, z+1), ind(x+1, y, z+1)])
        if (y < ny) and (z < nz):
            FV.append([h, ind(x, y+1, z), ind(x, y, z+1), ind(x, y+1, z+1)])

    return runComputation(hslice,idslice,bin_data,lock, nx, ny, nz,
         coloridx, calculateout, V, FV, datap,
                   BORDER_FILE, label) ⟨Run computation 20⟩
```

◇

Macro never referenced.

⟨Run computation 20⟩ ≡

```
def runComputation(hslice,idslice,bin_data, lock, imageDx, imageDy,
    imageDz, coloridx, calculateout,
    V, FV, datap, BORDER_FILE,  label):

    segmentation = datap['segmentation'].astype(np.uint8)

    segmentation[0, 0, 0] = 0
    segmentation[0, 0, 1] = 1
    datap['segmentation'] = segmentation

    logger.debug("unique %s " %(str(np.unique(datap['segmentation']))))
    imageHeight, imageWidth = datap['segmentation'][:,:,:].shape[1:3]

    imageDepth = datap['segmentation'].shape[0]
    Nx, Ny, Nz = imageHeight/imageDx, imageWidth/imageDx, imageDepth/imageDz

    returnValue = 2

    try:

        centroidsCalc = np.unique(datap['segmentation'])

        returnValue = startComputeChains(hslice,
                        idslice,bin_data,lock, imageHeight, imageWidth, imageDepth,
                        imageDx, imageDy, imageDz, Nx, Ny, Nz,
                        calculateout, BORDER_FILE,
                        centroidsCalc, coloridx,
                        datap) ⟨Start compute chains 21⟩
    except:
        exc_type, exc_value, exc_traceback = sys.exc_info()
        lines = traceback.format_exception(exc_type, exc_value, exc_traceback)
        log(1, [ "Error: " + ''.join('!! ' + line for line in lines) ])
        returnValue = 2

    return returnValue
```
◇

Macro referenced in 19b.

⟨Start compute chains 21⟩ ≡

```
def startComputeChains(hslice,idslice, bin_data,lock,
    imageHeight, imageWidth, imageDepth,
    imageDx, imageDy, imageDz,
    Nx, Ny, Nz, calculateout, BORDER_FILE,
    centroidsCalc, colorIdx, datap
):
    beginImageStack = 0
    endImage = beginImageStack
    saveTheColors = centroidsCalc
    log(2, [centroidsCalc])
    saveTheColors = np.array(
        sorted(saveTheColors.reshape(1, len(centroidsCalc))[0]), dtype=np.int)
    log(2, [saveTheColors])

    returnValue = 2

    threads = []
    for j in xrange(imageDepth / imageDz):
        startImage = endImage
        endImage = startImage + imageDz
        log(2, [ "Added task: " + str(j)
   + " -- (" + str(startImage) + "," + str(endImage) + ")" ])

      t1 = ChainsThreadComputation(hslice,idslice, bin_data,lock,
            startImage, endImage, imageHeight, imageWidth,
                                imageDx, imageDy, imageDz, Nx, Ny, Nz, calculateout,
                                BORDER_FILE, centroidsCalc,
                                colorIdx, datap) ⟨Chains thread computation 22⟩
      t1.start()
      threads.append(t1)

    for t in threads:
   t.join()

    log(1, [ "Completed: " + str(processRes) ])
    if (sum(processRes) == 0):
            returnValue = 0
    return returnValue
◇
```

Macro referenced in 20.

⟨ Chains thread computation 22 ⟩ ≡

```
class ChainsThreadComputation(threading.Thread):
   def __init__(self, hslice,idslice, bin_data,lock,
    startImage, endImage, imageHeight, imageWidth,
    imageDx, imageDy, imageDz,
    Nx, Ny, Nz,
    calculateout, BORDER_FILE,
    centroidsCalc, colorIdx, datap):
      threading.Thread.__init__(self)
      self.idslice=idslice
      self.bin_data=bin_data
      self.lock=lock
      self.hslice=hslice
      self.startImage=startImage
      self.endImage=endImage
      self.imageHeight=imageHeight
      self.imageWidth=imageWidth
      self.imageDx=imageDx
      self.imageDy=imageDy
      self.imageDz=imageDz
      self.Nx=Nx
      self.Ny=Ny
      self.Nz=Nz
      self.calculateout=calculateout
      self.BORDER_FILE=BORDER_FILE
      self.centroidsCalc=centroidsCalc
      self.colorIdx=colorIdx
      self.datap=datap

   def run(self):
       log(2, [ "Working task: " + str(self.startImage) + "-" + str(self.endImage) + " [" +
          str( self.imageHeight) + "-" + str( self.imageWidth ) + "-" + str(self.imageDx) +
          "-" + str( self.imageDy) + "-" + str (self.imageDz) + "]" ])
```

```
 bordo3 = None
 if (self.calculateout == True):
    with open(self.BORDER_FILE, "r") as file:
            bordo3_json = json.load(file)
            ROWCOUNT = bordo3_json['ROWCOUNT']
            COLCOUNT = bordo3_json['COLCOUNT']
            ROW = np.asarray(bordo3_json['ROW'], dtype=np.int32)
            COL = np.asarray(bordo3_json['COL'], dtype=np.int32)
            if np.isscalar(bordo3_json['DATA']):
                # in special case, when all numbers are same
                logger.debug('bordermatrix data stored as scalar 1')
                DATA = np.ones(COL.shape, dtype=np.int8) *\
                    np.int8(bordo3_json['DATA'])
            else:
                # this is general form
                logger.debug(
                    'bordermatrix data stored in general form')
                DATA = np.asarray(bordo3_json['DATA'], dtype=np.int8)
            bordo3 = csr_matrix(
                (DATA, COL, ROW), shape=(ROWCOUNT, COLCOUNT))

 xEnd, yEnd = 0, 0
 beginImageStack = 0
 saveTheColors = self.centroidsCalc
 saveTheColors = np.array(
sorted(saveTheColors.reshape(1, len(self.centroidsCalc))[0]), dtype=np.int
 )

 returnProcess = 0

 try:
try:
    log(2, ["Working task: " +
            str(self.startImage) + "-" +
            str(self.endImage) + " [loading colors]"])

    theImage = read_by_block(
        self.datap,
        self.startImage, self.endImage,
        self.centroidsCalc) ⟨Read by block 25a⟩

    log(2, ["Working task: " +
            str(self.startImage) + "-" +
            str(self.endImage) + " [comp loop]" ])
    for xBlock in xrange(self.imageHeight / self.imageDx):
        for yBlock in xrange(self.imageWidth/self.imageDy):
            xStart, yStart = xBlock * self.imageDx, yBlock * self.imageDy
            xEnd, yEnd = xStart+self.imageDx, yStart+self.imageDy

            image = theImage[:, xStart:xEnd, yStart:yEnd]

            nz, nx, ny = image.shape

            # Compute a quotient complex of chains with constant field
            # ------------------------------------------------------------

            chains3D_old = []
            chains3D = None
            hasSomeOne = False
            if (self.calculateout != True):
                chains3D = np.zeros(nx * ny * nz, dtype=np.int32)

            zStart = self.startImage - beginImageStack + self.hslice

            chains3D_old = cch.setList(
                    nx, ny, nz, self.colorIdx, image, saveTheColors) ⟨Set list 25c⟩

       # Compute the boundary complex of the quotient cell
            objectBoundaryChain = None
            if (self.calculateout == True) and (len(chains3D_old) > 0):
                objectBoundaryChain = larBoundaryChain(
                    bordo3, chains3D_old) #LAR

            # Save
            if (self.calculateout == True):
                if (objectBoundaryChain != None):
                    writeData(self.hslice,self.lock, ⟨Write data 25b⟩
                        self.bin_data,
                        np.array(
                            [zStart, xStart, yStart], dtype=int32),
                        objectBoundaryChain)

except:

    import traceback
    logger.debug(traceback.format_exc())
    exc_type, exc_value, exc_traceback = sys.exc_info()
    lines = traceback.format_exception(
        exc_type, exc_value, exc_traceback)
    # Log it or whatever here
    log(1, ["Error: " + ’’.join(’!! ’ + line for line in lines)])
```

```
          returnProcess = 2

       except:
      import traceback
      exc_type, exc_value, exc_traceback = sys.exc_info()
      print sys.exc_info()
      lines = traceback.format_exception(exc_type, exc_value, exc_traceback)
      log(1, ["Error: " + ''.join('!! ' + line for line in lines)])
      returnProcess = 2
```
◇

Macro referenced in 21.

⟨ Read by block 25a ⟩ ≡

```
def read_by_block(datap, startImage, endImage, centroidsCalc):
    segmentation = datap['segmentation'][startImage:endImage:, :, :]
    return segmentation
```
◇

Macro referenced in 22.

⟨ Write data 25b ⟩ ≡

```
def writeData(hslice,lock,bin_data,offsetCurr, objectBoundaryChain):
    lock.acquire()
    bin_data.append([[offsetCurr[0],offsetCurr[1],
    offsetCurr[2]],objectBoundaryChain.toarray()])
    lock.release()
```
◇

Macro referenced in 22.

11 Calculate chains helper

⟨ Set list 25c ⟩ ≡

```
def setList(int nx, int ny, int nz, int colorIdx, np.ndarray[np.uint8_t, ndim=3] image,
   np.ndarray[np.int_t, ndim=1] saveTheColors):
   cdef list chains3D_old = range(0)

   for x in xrange(nx):
      for y in xrange(ny):
         for z in xrange(nz):
            if (image[z,x,y] == saveTheColors[colorIdx]):
               chains3D_old.append(addr(x,y,z,nx,ny,nz)) ⟨ Addr 26a ⟩

   return chains3D_old
```
◇

Macro referenced in 22.

⟨ Addr 26a ⟩ ≡

```
cdef int addr(int x, int y, int z, int nx, int ny, int nz) nogil:
   return x + (nx) * (y + (ny) * (z))
⋄
```

Macro referenced in 25c.

12 Generate a square mesh

⟨ Square mesh 26b ⟩ ≡

```
def square_mesh(nx, ny, nz,bin_data):

    def ind(x,y,z): return x + (nx+1) * (y + (ny+1) * (z))

    chunksize = nx * ny + nx * nz + ny * nz + 3 * nx * ny * nz
    V = [[x,y,z] for z in xrange(nz+1) for y in xrange(ny+1) for x in xrange(nx+1) ]

    v2coords = invertIndex(nx,ny,nz) ⟨ Invert index 18 ⟩

    FV = []
    for h in xrange(len(V)):
        x,y,z = v2coords(h)
        if (x < nx) and (y < ny): FV.append([h,ind(x+1,y,z),ind(x,y+1,z),ind(x+1,y+1,z)])
        if (x < nx) and (z < nz): FV.append([h,ind(x+1,y,z),ind(x,y,z+1),ind(x+1,y,z+1)])
        if (y < ny) and (z < nz): FV.append([h,ind(x,y+1,z),ind(x,y,z+1),ind(x,y+1,z+1)])

    logger.debug(’before readFile()’)
    try:
        V2,F2=read(V,FV,bin_data,chunksize) ⟨ Read 26c ⟩
    except:
        import traceback
        traceback.print_exc()
        exc_type, exc_value, exc_traceback = sys.exc_info()
        lines = traceback.format_exception(exc_type, exc_value, exc_traceback)
        log(1, [ "Error: " + ’’.join(’!! ’ + line for line in lines) ])
        sys.exit(2)
    logger.debug(’after readFile()’)
    return V2,F2
⋄
```

Macro never referenced.

⟨ Read 26c ⟩ ≡

```
def read(V,FV,bin_data,chunksize):
    V2=[]
    F2=[]
    vertex_count = 1
    old_vertex_count = vertex_count
    count = 0
    try:
      for indice in range(len(bin_data)):
         count += 1
      zStart = bin_data[indice][0][0]
                xStart = bin_data[indice][0][1]
                yStart = bin_data[indice][0][2]
                log(1, ["zStart, xStart, yStart = "
      + str(zStart) + "," + str(xStart) + "," + str(yStart)]);
      LISTA_VETTORI2=bin_data[indice][1]
                lista = LISTA_VETTORI2
                LISTA_VETTORI2 = np.abs(LISTA_VETTORI2)
                timer_stop();
                timer_start("objectBoundaryChain ");
                l = len(LISTA_VETTORI2)
                objectBoundaryChain = scipy.sparse.csr_matrix(LISTA_VETTORI2.reshape((l,1)))
                timer_stop();
                b2cells = csrChainToCellList(objectBoundaryChain) #LAR
                FVn = []
                for i, face in enumerate(FV):
                  [v1, v2, v3, v4] = FV[i]
                            # face = [v1, v2, v4, v3]
                        if lista[i] < 0:
                           FVn.append([v1, v3, v2, v4])
                        else:
                                FVn.append([v1, v2, v3, v4])
                vertex_count, old_vertex_count = write(
                            V2, F2,
                            V, FVn,
                            xStart, yStart, zStart,
                            vertex_count, old_vertex_count,
                            b2cells
                        ) ⟨Write 28a⟩
    except struct.error:
      logger.debug('not importatnt reading error')
    except:
      logger.debug('reading error')
        traceback.print_exc()
        exc_type, exc_value, exc_traceback = sys.exc_info()
         lines = traceback.format_exception(exc_type, exc_value, exc_traceback)
        log(1, [ "EOF or error: " + ’’.join(’!! ’ + line for line in lines) ])
    return V2,F2
◇
```

Macro referenced in 26b.

⟨ Write 28a ⟩ ≡

```
def write(V2, F2, V, FV,
                       xStart, yStart, zStart,
                       vertex_count, old_vertex_count,
                       b2cells
                       ):
    for f in b2cells:
        old_vertex_count = vertex_count

        for vtx in FV[f]:
            x=V[vtx][0] + xStart
            y=V[vtx][1] + yStart
            z=V[vtx][2] + zStart
            V2.append([x,y,z])
            vertex_count = vertex_count + 1

        F2.append([
            old_vertex_count + 0,
            old_vertex_count + 1,
            old_vertex_count + 3,
            old_vertex_count + 2
        ])
    return vertex_count, old_vertex_count
◇
```

Macro referenced in 26c.

13 Cleaning and smoothing

⟨ Make cleaning and smoothing 28b ⟩ ≡

```
def makeCleaningAndSmoothing(V, F):
    V, F = rmbox.removeDoubleVertexesAndFaces(V, F, use_dict_algorithm=False)
    V = ls.makeSmoothing(V, F) ⟨ Make smoothing ? ⟩
    V, F = rmbox.removeDoubleVertexesAndFaces(V, F, use_dict_algorithm=False)
    return V, F
◇
```

Macro never referenced.

14 Visus+LAR: C++ implementation

Aim of this document is to define the architecture of the C++ implementation of the LAR+Visus application, starting from the existing python prototype.

Query and data preparation on the upper slab

- Load the parameters of computation from the configuration file;
- the visus query on the *idx* file extracts slabs of 3D venous/neural data at finest resolution as a three-dimensional row-major matrix;
- application of a threshold on the data;
- removal of the small connected components;
- computation of the boundary operator;
- at a certain time three adjacent slabs of the volume are computed by different processes, as shown in figure.

Model generation on the central slab

- The central slab of the three in memory is divided into bricks;
- boundary extraction within every brick through an SpMV multiplication between $[\partial_3]$ and the 3-chain;
- double vertices and faces removal;
- computation of the indices for the crop of the cover;
- laplacian or taubin non-shrinking smoothing;
- crop of the cover;
- topology-preserving simplification.

Writing on the lower slab

- At the end of each slab computation, each process writes verteces and faces on the same obj file.

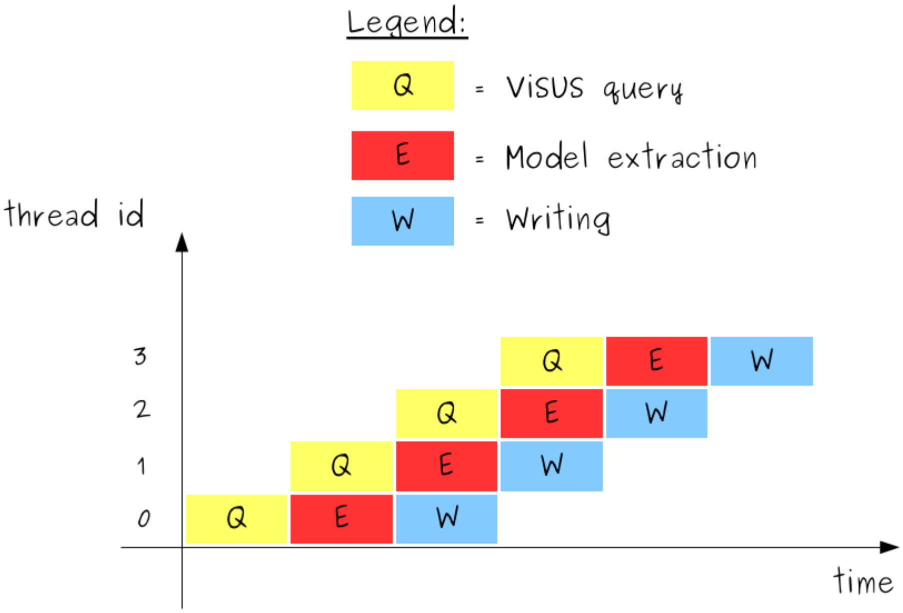

Figure 6: *Gant diagram of the computation.*

Visualization of the lower slab

- Visualization of the first slab through the $VisusViewer$;
- Update the visualization each time a new slab is ready.

References

A. PAOLUZZI, A. DI CARLO, F. FURIANI, M. JIRIK, *CAD models from medical images using LAR*, Computer-Aided Design and Applications,2015. Preliminary version in CAD'15, June 22-25, 2015, London, UK;

F. FURIANI, C. PAOLUZZI, A. PAOLUZZI, *Algebraic extraction of models and properties from images* (in Italian), GeoMedia, Volume 17, Issue 6, December 2013.