Geometric model extraction from 3D medical data [*]

Giulia Clementi, CVD Lab, Roma Tre University

November 24, 2015

Abstract

Technological advances made it possible to acquire massive sets of 3D biomedical data. The collected knowledge has to be formalized, organized and combined in many ways. Furthermore, simulations and interactive explorations are needed. This document is aimed to describe model extraction from 3D medical images and visualization using the LAR framework.

Contents

[*] This document is part of the *Linear Algebraic Representation with CoChains* (LAR-CC) framework. November 24, 2015

1 Introduction to LarVolumeToObj

LarVolumeToObj is a software module of the LAR(Linear Algebraic Representation) framework designed to generate a well-defined 3D mesh from a stack of 2D images. It provides a boundary model for both the relevant features, such as surfaces, and their outer space. Generally speaking, the structure of a d-dimensional image is mapped to a cellular complex of d-cuboids or voxels.

1.1 LarVolumeToObj software architecture

LarVolumeToObj is structured into three major components: those involved in data preparation, model generation and data visualization.

Data preparation

- memorization of 3D data, a stack of 2D images on a three-dimensional matrix;
- denoising via median filter;
- color quantization via K-means clustering;
- use of pklz, where pklz is a python object serializator;
- segmentation of the pklz.

Model generation

- generation of matrix $[\ \partial_3\ ]$ (boundary operator);
- boundary chain computation through a SpMV multiplication;
- double facets removal via a map-reduce algorithm;
- laplacian smoothing;
- triangulation of the quads;
- use of json and obj formats (currently).

Visualization layer

- model load;
- lar or pyplasm visualization.

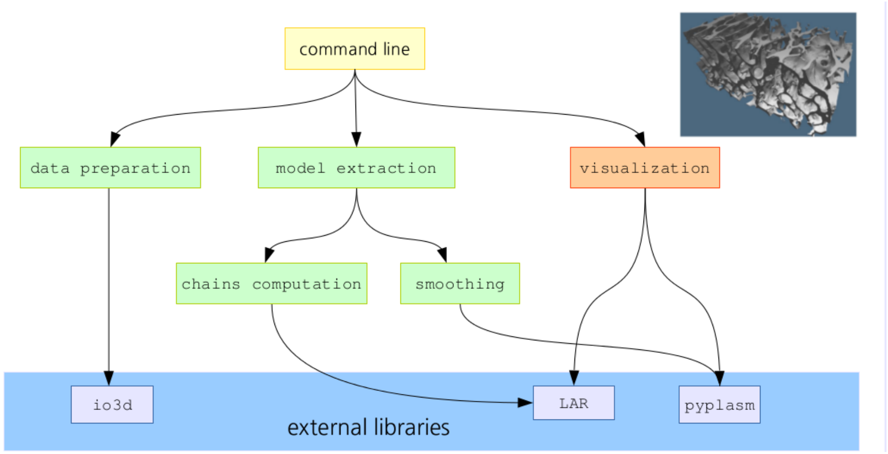

Figure 1: LarVolumeToObj software architecture.

1.2 Run a basic example

⟨Basic example 4a⟩ ≡

```
import larVolumeToObj
larVolumeToObj.computation.data_preparation.preparedata
("./biodur_sample/", 'biodur_crop.pklz',
crop=[[1, 25], [200, 225], [200, 225]], threshold=1400) ⟨Crop and threshold 4b⟩
V, F = larVolumeToObj.computation.pklzToSmoothObj.makeSmooth
('biodur_crop.pklz', bordersize=[5,5,5]) ⟨Make smooth 6⟩
larVolumeToObj.computation.visualization.visualize(V, F, explode=False) ⟨Visualize 29⟩
```

◇

Macro never referenced.

2 Data preparation

At first, data preparation is needed. Input data, a set of 2D images, need to be cropped. Then a threshold has to be applyed. Crop is about selecting from the data a cubic portion to be modeled and cutting out the other parts. Crop parameters have to be specified in the format *[[minX, maxX], [minY, maxY], [minZ, maxZ]]*. The threshold permits to decide which pixels are relevant and belong to the model and which not. Pixels whose intensity is under the threshold have not to be considered. After these two operations, the selected data are saved in pklz format, where pklz is a python object serializator. To read and write the data is used an external library from the same authors of LarVolumeToObj called io3d.

⟨Crop and threshold 4b⟩ ≡

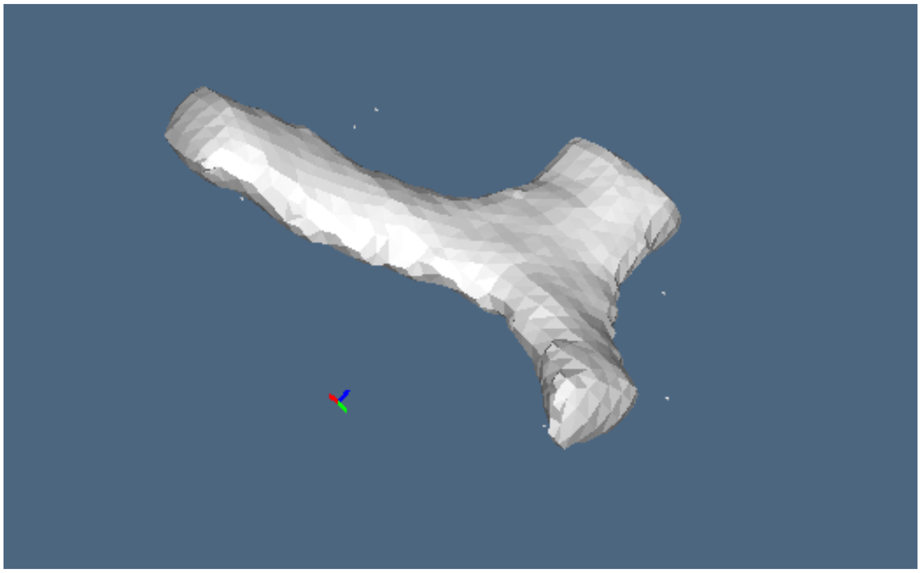

Figure 2: Visualization of a small data sample from micro-CT image of pig liver corrosion cast prepared in faculty hospital in Pilsen (Czech Republic).

```
def preparedata(inputfile, outputfile=’prepared.pklz’, crop=None, threshold=None,
visualization=False, zero_border=0, label=2):
  datap = io3d.datareader.read(inputfile, dataplus_format=True)
  if crop is not None:
    datap[’data3d’] = datap[’data3d’][crop[0][0]:crop[0][1],
    crop[1][0]:crop[1][1], crop[2][0]:crop[2][1]]
      if ’segmentation’ in datap.keys():
        datap[’segmentation’] = datap[’segmentation’]
        [crop[0][0]:crop[0][1], crop[1][0]:crop[1][1], crop[2][0]:crop[2][1]]
  if threshold is not None:
     datap[’segmentation’] = (datap[’data3d’] > threshold).astype(np.uint8) * label
  if visualization:
     ed = sed3.sed3(datap[’data3d’], contour=datap[’segmentation’])
     ed.show()
  if zero_border > 0:
     datap[’segmentation’][:zero_border, :, :] = 0
     datap[’segmentation’][:, :zero_border, :] = 0
     datap[’segmentation’][:, :, :zero_border] = 0
     datap[’segmentation’][-zero_border:, :, :] = 0
     datap[’segmentation’][:, -zero_border:, :] = 0
     datap[’segmentation’][:, :, -zero_border:] = 0
io3d.datawriter.write(datap, outputfile)
```

◇

Macro referenced in 4a.

3 From pklz to smooth obj

The main contribution of LAR is the idea of a simple and general representation, that offers the possibility to model topological data. The aim of this section is the computation of the geometric model and its smoothing. The geometric model, extracted from input file, pklz, is saved in obj format after its smoothing.

⟨Make smooth 6⟩ ≡

```
def makeSmooth(
    inputfile,
    bordersize=[2, 2, 2],
    outputdir=’output’,
    outputfile=’out’,
    visualization=False,
    borderdir=’border’,
    make_triangulation=True,
    label=2
):
    filepath, ext = os.path.splitext(inputfile)
    if ext == "obj":
        obj_input = inputfile
    else:
        print ’Processing pklz data’
        convert(inputfile, bordersize, outputdir, borderdir=borderdir,
        label=label) ⟨Convert pklz to obj 7⟩
        obj_input = ’stl/model-2.obj’
    V, F = readFile(os.path.join(outputdir, obj_input)) ⟨read file 9⟩
    #read the obj
    print "Before"
    print "Number of vertexes: %i    Number of faces %i" % (len(V), len(F))
    V, F = makeCleaningAndSmoothing(
    V, F, os.path.join(outputdir, outputfile)) ⟨Smooth and clean 8⟩
    print "After"
    print "Number of vertexes: %i    Number of faces %i" % (len(V), len(F))
    # write to ints
    # fill empty vertexes
    V = [v if len(v) == 3 else [0, 0, 0] for v in V]
    # make tenimes bigger
    Vint = (np.asarray(V) * 10).astype(np.int).tolist()
    if outputfile is not None:
        writeFile(
        os.path.join(outputdir, outputfile + "_sm_i.obj"),
        Vint, F, ignore_empty_vertex_warning=True) ⟨ write file 12 ⟩
    #write smoothed model in obj file
    # make triangulation
    if make_triangulation:
        Ftr = save_triangulated(V, Vint, F, outputdir, outputfile)
    if visualization:
        Ftr = triangulate_quads(F)
        vis.visualize(V, Ftr)
    return V, F
```

◇

Macro referenced in 4a.

3.1 Convert pklz to obj

To convert pklz to obj *nx, ny* and *nz* are needed. These variables represent the three dimensions of the data. The corresponding boundary operator matrix has to be generated by LAR and saved in a json file. The json file is then given to the module for the computation of boundary cells. Cochains are saved at first in some binary files, then unified in a unique binary file to be translated and moved in an obj file.

⟨ Convert pklz to obj 7 ⟩ ≡

```
def convert(
    filename,
    bordersize=[2, 2, 2],
    outputdir=’tmp/output’,
    borderdir=’./tmp/border’,
    label=2
):

    bindir = os.path.join(outputdir, ’compbin’)
    stldir = os.path.join(outputdir, ’stl’)
    binfile = os.path.join(bindir, ’model-2.bin’)
    stlfile = os.path.join(stldir, ’model-2.obj’)

    if os.path.exists(outputdir):
        shutil.rmtree(outputdir)
    mkdir_p(outputdir)
    mkdir_p(stldir)
    mkdir_p(bindir)
    mkdir_p(borderdir)

    nx, ny, nz = bordersize
    brodo3path = gbmatrix.getOrientedBordo3Path(nx, ny, nz, borderdir)
    ⟨Get border matrix 23a⟩
    logger.debug("in convert()")

    # read data per blocks, find boundary, write each block to binary file
    s2bin.calcchains_main(
        nx=nx, ny=ny, nz=nz,
        calculateout=True,
        input_filename=filename,
        # BORDER_FILE='./tmp/border/bordo3_2-2-2.json',
        BORDER_FILE=brodo3path,
        # BORDER_FILE=border_file,
        DIR_O=bindir,
        # colors=,
        coloridx=label,
        label=label
        )  ⟨Compute chains 15b⟩
    logger.debug("calcchains_main finished")
    #unify binary files in a unique binary
    concatenate_files(
        bindir + "/*.bin",
        binfile)
    logger.debug("concatenate() finished")
    # nx, ny, nz = boxsize
    #create obj from binary file
    sq.make_obj(
        nx, ny, nz,
        binfile,
        stldir) ⟨Make obj 26c⟩
    logger.debug("obj file  finished")

    concatenate_files(stldir + '/output-*-*.stl', stlfile)
```

◇

Macro referenced in 6.

3.2 Smooth and clean

Since double vertices and faces have been introduced from parallel processes, that have computed boundary chains, an important step is to remove them.

⟨Smooth and clean 8⟩ ≡

```
def makeCleaningAndSmoothing(V, F, outputfile=None):
    logger.debug("outputfile " + str(outputfile))
    # findBoxVertexesForAxis(v, 2, 0)
    # v, f = findBoundaryFaces(v, f, 2)

    # TODO debug dict algorithm
    V, F = rmbox.removeDoubleVertexesAndFaces(V, F, use_dict_algorithm=False)
    ⟨remove double vertices and faces 13b⟩
    if outputfile is not None:
        writeFile(outputfile + "_cl.obj", V, F)
    V = ls.makeSmoothing(V, F) ⟨smooth 25b⟩
    # TODO remove unused vertexes is too general and slow
    V, F = rmbox.removeDoubleVertexesAndFaces(V, F, use_dict_algorithm=False)
    ⟨remove double vertices and faces 13b⟩
    V, F = __remove_first_vertex(V, F)

    if outputfile is not None:
        writeFile(outputfile + "_sm.obj", V, F,
        ignore_empty_vertex_warning=True)
    return V, F
◇
```

Macro referenced in 6.

4 File io

This file contains support primitives to the step of model extraction from pklz to obj.

⟨read file 9⟩ ≡

```
def readFile(filename, ftype='auto', shift_obj=True):
    if filename.find('*') == -1 and \
        filename.find('?') == -1 and \
            filename.find('[') == -1:

        V, F = readOneFile(filename, ftype=ftype, shift_obj=shift_obj) ⟨read one file 10⟩

    else:
        V = []
        F = []
        filelist = glob.glob(filename)
        for fl in filelist:
            V1, F1 = readOneFile(fl, ftype=ftype, shift_obj=shift_obj)
            lenV = len(V)
            V = V + V1
            # add  len of prev V to indexes
            F1 = (np.array(F1) + lenV).tolist()
            F = F + F1
            # import ipdb; ipdb.set_trace() #  noqa BREAKPOINT

    return V, F
```
◇

Macro referenced in 6.

4.1 Read a single file

This method can read many formats including:

- Pickle format: in order to de-serialize pkl file and reconstructing and returning the original object hierarchy, is used the pickle module and its function *load*. It reads a string from the open file object and interprets it as a pklz data stream. Vertices and faces are returned.

- Obj format: to load the obj file and its faces and vertices is called the function *readObjStream*.

⟨ read one file 10 ⟩ ≡

```
def readOneFile(filename, ftype='auto', shift_obj=True):
    """
    File open one file. It can autodetect type.
    """
    if not os.path.isfile(filename):
        logger.error('File "%s" not found' % (filename))
        exit(2)
    if ftype == 'auto':
        _, ext = os.path.splitext(filename)
        ftype = ext[1:]
        logger.info('file type autodetect: ' + ftype)

    if ftype in ('pkl', 'pickle', 'p'):
        fdata = pickle.load(open(filename, "rb"))
        vertexes, faces = fdata
    elif ftype in ('rawc'):
        with open(filename, "r") as f:
            vertexes, faces = __readRawcStream(f)
    elif ftype == 'obj':
        with open(filename, "r") as f:
            vertexes, faces = __readObjStream(f) ⟨ read obj stream 11b ⟩
        if shift_obj:
            faces = (np.asarray(faces) - 1).tolist()

    return vertexes, faces
◇
```

Macro referenced in 9.

4.2 Read obj stream

Here the aim is to extract the array of vertices and the array of faces from an obj stream and return them, according with LAR. Obj file contains the unordered list of vertices and faces in the format:

⟨sample obj 11a⟩ ≡

```
# List of geometric vertices, with (x,y,z[,w]) coordinates
# w is the homogeneous coordinate, default 1.
v 250 50 60
...
# Polygonal face elements. Faces are defined using vertex indices.
f 263 342 340 256
◇
```

Macro never referenced.

⟨read obj stream 11b⟩ ≡

```
def __readObjStream(f):
   vertexes = []
    faces = []
    for line in f.readlines():
        lnarr = line.strip().split(' ')
        if lnarr[0] == 'v':
            try:
                vertex = [
                    int(lnarr[1]),
                    int(lnarr[2]),
                    int(lnarr[3])
                ]
            except ValueError:
                vertex = [
                    float(lnarr[1]),
                    float(lnarr[2]),
                    float(lnarr[3])
                ]

            vertexes.append(vertex)
        elif lnarr[0] == 'f':
            face = [0] * (len(lnarr) - 1)
            for i in range(1, len(lnarr)):
                face[i - 1] = int(lnarr[i])
            faces.append(face)

    return vertexes, faces
```
◇

Macro referenced in 10.

4.3 Write an obj file

⟨ write file 12 ⟩ ≡

```
def writeFile(filename, vertexes, faces, ftype='auto', shift_obj=True,
              ignore_empty_vertex_warning=False):
    """
    filename
    vertexes
    faces
    """
    if ftype == 'auto':
        _, ext = os.path.splitext(filename)
        ftype = ext[1:]

    if ftype in ('pkl', 'pickle', 'p'):
        pickle.dump([vertexes, faces], open(filename, 'wb'))
    elif ftype == 'obj':
        if shift_obj:
            faces = (np.asarray(faces) + 1).tolist()
        with open(filename, "w") as f:
            for i, vertex in enumerate(vertexes):
                __writeVertexLineToObjFile(f, vertex,
                                           ignore_empty_vertex_warning)
      ⟨ write vertex to obj file 13a ⟩
                # import ipdb; ipdb.set_trace() #  noqa BREAKPOINT

            for face in faces:
                fstr = "f"
                for i in range(0, len(face)):
                    fstr += " %i" % (face[i])

                fstr += "\n"

                f.write(fstr)
```
◇

Macro referenced in 6.

⟨write vertex to obj file 13a⟩ ≡

```
def __writeVertexLineToObjFile(f, vertex, ignore_empty_vertex_warning):
    try:
        f.write("v %s %s %s\n" % (
            str(vertex[0]),
            str(vertex[1]),
            str(vertex[2])
        ))
    except IndexError:
        if not ignore_empty_vertex_warning:
            logger.warning('empty vertex %i ' % (vertex))
        f.write("v 0 0 0\n")
◇
```

Macro referenced in 12.

5 Remove from boxes inner faces

Here is presented a *map-reduce* algorithm to achieve double facets removal.

⟨remove double vertices and faces 13b⟩ ≡

```
def removeDoubleVertexesAndFaces(vertexes, faces, boxsize=None, index_base=0,
                                 use_dict_algorithm=True):
    """
    Main function of module. Return object description cleand from double
    vertexes and faces.
    """

    t0 = time.time()
    # new_vertexes, inv_vertexes = removeDoubleVertexes(vertexes)
    new_vertexes, inv_vertexes = removeDoubleVertexesAlternative(vertexes)
   ⟨remove double vertices 14a⟩
    t1 = time.time()
    logger.info("Doubled vertex removed          " + str(t1 - t0))
    logger.info("Number of vertexes: %i " % (len(new_vertexes)))
    new_faces = reindexVertexesInFaces(faces, inv_vertexes,
                                       index_base=index_base) ⟨reindex vertices 14b⟩
    t2 = time.time()
    logger.info("Vertexes in faces reindexed     " + str(t2 - t1))
    if use_dict_algorithm:
        logger.debug("Using dict algotithm by Alberto")
        new_faces = removeDoubleFaces(new_faces) ⟨remove double faces 15a⟩
    else:
        if boxsize is None:
            logger.debug("Using removeDoubleFaces")
            new_faces = removeDoubleFaces(new_faces)
        else:
            logger.debug("Using removeDoubleFacesOnlyOnBoundaryBoxes")
            new_faces = removeDoubleFacesOnlyOnBoundaryBoxes(
                new_vertexes, new_faces, boxsize[0], index_base=index_base)
    t3 = time.time()
    logger.info("Double faces removed            " + str(t3 - t2))
    return new_vertexes, new_faces
◇
```

Macro referenced in 8.

5.1 Double vertices removal

⟨ remove double vertices 14a ⟩ ≡

```
def removeDoubleVertexesAlternative(V):
    X = range(len(V))
    Vs = []
    Is = [0]*len(V)

    prevv = None
    i = 0
    for [v, x] in sorted(zip(V, X)):
        if v == prevv:
            # prev index was increased
            Is[x] = i - 1
        else:
            Vs.append(v)
            Is[x] = i
            i = i + 1
            prevv = v

    return Vs, Is
◇
```

Macro referenced in 13b.

5.2 Vertices reindicisation

⟨ reindex vertices 14b ⟩ ≡

```

def reindexVertexesInFaces(faces, new_indexes, index_base=0):
    for face in faces:
        try:
            for i in range(0, len(face)):
                face[i] = new_indexes[face[i] - index_base] + index_base
        except:
            import traceback
            traceback.print_exc()
            print 'fc ', face, ' i ', i
            print len(new_indexes)

    return faces
◇
```

Macro referenced in 13b.

5.3 Double faces removal

⟨ remove double faces 15a ⟩ ≡

```
def removeDoubleFaces(faces):
    from collections import defaultdict
    cellDict = defaultdict(list)
    for k, cell in enumerate(FW):
        cellDict[tuple(cell)] += [k]
    FW = [list(key) for key in cellDict.keys() if len(cellDict[key]) == 1]
    return FW
◇
```

Macro referenced in 13b.

6 Chains computation

⟨ Compute chains 15b ⟩ ≡

```
def calcchains_main(
    nx, ny, nz,
    calculateout,
    input_filename,
    BORDER_FILE,
    DIR_O,
    # colors,
    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)
    # mj
    # 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(nx, ny, nz, coloridx, calculateout, V, FV, input_filename,
                   BORDER_FILE, DIR_O, label) ⟨ Chains computation 16 ⟩
```

◇

Macro referenced in 7.

6.1 Parallel computation

This parallel function creates a pool of processes with *multiprocessing.Pool*. The process pool object controls worker processes to which jobs are submitted. The number of worker processes to use can be chosen. Chains computation is done in parallel by extracting at the same time the boundary chain from different segments of the data. Focus on the function that computes chains: its parameters are the json file with the boundary operator matrix, information about segmentation and clusterization of the data. In bin files are written from parallel processes double vertices and faces. Double vertices and faces removal is done during the next step, smoothing.

⟨ Chains computation 16 ⟩ ≡

```
def startComputeChains(
    imageHeight, imageWidth, imageDepth,
    imageDx, imageDy, imageDz,
    Nx, Ny, Nz, calculateout, BORDER_FILE,
    centroidsCalc, colorIdx, datap, DIR_O
):

    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

 processPool = max(1, multiprocessing.cpu_count() / 2)
 #Return the number of CPUs in the system/2 or 1
 log(2, ["Starting pool with: " + str(processPool)])

 try:
     pool = multiprocessing.Pool(processPool)
     log(2, ['Start pool'])

#computing segmentation of the data
     for j in xrange(imageDepth / imageDz):
         startImage = endImage
         endImage = startImage + imageDz
         log(2, [ "Added task: " + str(j) + " --
         (" + str(startImage) + "," + str(endImage) + ")" ])

         pool.apply_async(
                         computeChainsThread,
                         args = (startImage, endImage, imageHeight, imageWidth,
                             imageDx, imageDy, imageDz, Nx, Ny, Nz, calculateout,
                             BORDER_FILE, centroidsCalc,
                             colorIdx, datap, DIR_O, ),
                         callback = collectResult) ⟨Compute chains process 18⟩
#are given to the function information about segmantation
and the file that contains boundary operator
     log(2, [ "Waiting for completion..." ])
     pool.close() #Prevents any more tasks from being submitted to the pool.
     Once all the tasks have been completed the worker processes will exit.
     pool.join() #Wait for the worker processes to exit.

     log(1, [ "Completed: " + str(processRes) ])
     if (sum(processRes) == 0):
         returnValue = 0
  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) ])
     # Log it or whatever here

 return returnValue
```

◇

Macro referenced in 15b.

6.2 Chain computation process

At first, boundary operator is loaded. The extraction of the segment to process is done by use of support function *readpklbyblock* and via an iteration on x and y coordinates. Every process partitions the data in quotient groups based on intensity values of pixels and computes the oriented boundary chain. The chain is written in a binary file, different for each stack partition of images. For each segment is saved in binary file the chain, computed with LAR, and the offset of the segment, represented by the variables *xstart, ystart, zstart.*

⟨ Compute chains process 18 ⟩ ≡

```
def computeChainsThread(
    startImage, endImage, imageHeight, imageWidth,
    imageDx, imageDy, imageDz,
    Nx, Ny, Nz,
    calculateout, BORDER_FILE,
    centroidsCalc, colorIdx, datap, DIR_O
):
    # centroidsCalc - remove
    # TODO use centroidsCalc
    # print 'cC '
    # print centroidsCalc

    # centroidsCalc = np.array([[0],[ 1]])
    log(2, [ "Working task: " + str(startImage) + "-" + str(endImage) + " [" +
            str( imageHeight) + "-" + str( imageWidth ) + "-" + str(imageDx) +
            "-" + str( imageDy) + "-" + str (imageDz) + "]" ])

    #Load border operator matrix
    bordo3 = None
    if (calculateout == True):
            with open(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)
                # print "border m ",  ROW.shape, COL.shape, DATA.shape
                # print  "5555555555555555555555555555555555555"
                bordo3 = csr_matrix(
                    (DATA, COL, ROW), shape=(ROWCOUNT, COLCOUNT))

xEnd, yEnd = 0, 0
beginImageStack = 0
# TODO do something with the input colorNumber
saveTheColors = centroidsCalc
# saveTheColors = np.array([1,0])
saveTheColors = np.array(
    sorted(saveTheColors.reshape(1, len(centroidsCalc))[0]), dtype=np.int
)
#prepare binary file for output
fileName = "pselettori-"
if (calculateout == True):
        fileName = "poutput-"
fileName = fileName + str(startImage) + "_" + str(endImage) + "-"

returnProcess = 0

try:
    fullfilename = DIR_O + '/' +\
        fileName + str(saveTheColors[colorIdx]) + BIN_EXTENSION
    logger.debug("file to write " + fullfilename)
    fileToWrite = open(fullfilename, "wb")
    try:
        log(2, ["Working task: " +
                str(startImage) + "-" +
                str(endImage) + " [loading colors]"])
        # theImage,colorNumber,theColors = pngstack2array3d(
        #        imageDir, startImage, endImage, colorNumber,
        #        pixelCalc, centroidsCalc)

        # imageDirPkl = "data.pklz"
    # -------------------------------
   #select the stack of images that contains the segment to be examined
        theImage = read_pkl_by_block(
            datap,
            startImage, endImage,
         centroidsCalc) ⟨read pkl by block 22a⟩
     # colorIdx = 2
     # print "orig shape ", datap['segmentation'].shape
     # print "png stack"
     # print 'startim ', startImage
     # print 'endim', endImage
     # print 'unique', np.unique(theImage)
     # print 'centrCol ', centroidsCalc
     # print 'saveTheColors ', saveTheColors, colorIdx
     # print 'calculateout ', calculateout
     # import ipdb; ipdb.set_trace() #  noqa BREAKPOINT
     # theColors = theColors.reshape(1,colorNumber)
     # if (sorted(theColors[0]) != saveTheColors):
     #    log(1, [ "Error: colorNumber have changed"] )
     #    sys.exit(2)

     log(2, ["Working task: " +
             str(startImage) + "-" +
             str(endImage) + " [comp loop]" ])
#iterate on x and y to select the segment
     for xBlock in xrange(imageHeight / imageDx):
         for yBlock in xrange(imageWidth/imageDy):
             xStart, yStart = xBlock * imageDx, yBlock * imageDy
             xEnd, yEnd = xStart+imageDx, yStart+imageDy

             image = theImage[:, xStart:xEnd, yStart:yEnd]
             # print "image ", image
             nz, nx, ny = image.shape

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

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

             zStart = startImage - beginImageStack

             if (calculateout == True):
                 chains3D_old = cch.setList(
                     nx, ny, nz, colorIdx, image, saveTheColors) ⟨set list 22b⟩
             else:
                 hasSomeOne, chains3D = cch.setListNP(
                     nx, ny, nz, colorIdx, image, saveTheColors)
```

```
            # Compute the boundary complex of the quotient cell
  # Use lar
            objectBoundaryChain = None
            if (calculateout == True) and (len(chains3D_old) > 0):
                objectBoundaryChain = larBoundaryChain(
                    bordo3, chains3D_old)
            #give for parameter boundary operator
            #and the chain complex belonging to the model

            # print objectBoundaryChain
            # brd = bordo3.todense()
            # print "chains3D_old"
            # print chains3D_old
            # print len(chains3D_old)
            # print "objectBoundaryChain s",
            # if objectBoundaryChain is not None:
            #     # print objectBoundaryChain
            #     print "e ", objectBoundaryChain.todense().shape
            #     print objectBoundaryChain.toarray().astype(’b’).flatten()
            # Save
            if (calculateout == True):
                if (objectBoundaryChain != None):
                    writeDataToFile(
                        fileToWrite,
                        np.array(
                            [zStart, xStart, yStart], dtype=int32),
                        objectBoundaryChain)
            else:
                if (hasSomeOne != False):
                    writeOffsetToFile(
                        fileToWrite,
                        np.array([zStart, xStart, yStart], dtype=int32)
                    )
                    fileToWrite.write(
                        bytearray(np.array(
                            chains3D, dtype=np.dtype(’b’))))
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
        finally:
            fileToWrite.close()
    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)])
        returnProcess = 2

    return returnProcess
```

◇

Macro referenced in 16.

6.3 Read pkl by block

Select a stack of images from *startImage* to *endImage*.

⟨ read pkl by block 22a ⟩ ≡

```
def read_pkl_by_block(datap, startImage, endImage, centroidsCalc):
    segmentation = datap[’segmentation’][startImage:endImage:, :, :]
    return segmentation
```

◇

Macro referenced in 18.

7 Calculate chain helper

This file contains support methods. *Set list* is a condition to verify if each pixel of a given segment belongs or not to the model.

⟨ set list 22b ⟩ ≡

```
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))

   return chains3D_old
```

◇

Macro referenced in 18.

8 Generate border matrix

Generate the boundary operator with LAR and save it in a json file.

⟨ Get border matrix 23a ⟩ ≡

```
def getOrientedBordo3Path(nx, ny, nz, DIR_OUT):
    fileName = DIR_OUT+'/bordo3_'+str(nx)+'-'+str(ny)+'-'+str(nz)+'.json'
    V, bases = ⟨ Get bases 23c ⟩
    VV, EV, FV, CV = bases
    brodo3 = computeOrientedBordo3(nx, ny, nz) ⟨ Compute oriented border 25a ⟩
    writeBordo3(brodo3, fileName) ⟨ write boundary operator 23b ⟩ #write in json file
    return fileName
◇
```

Macro referenced in 7.

⟨ write boundary operator 23b ⟩ ≡

```
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 23a.

8.1 Get bases

This function generates a 3D space of total dimensions *nx, ny, nz*. This space is made by unit cubes.

⟨ Get bases 23c ⟩ ≡

```
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)

    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, FV, CV, VV, EV
```

```
        return V, (VV, EV, FV, CV)
    ◇
```

Macro referenced in 23a, 25a.

8.2 Compute oriented border

Use of lar-cc software framework to calculate the boundary operator matrix given a bases for the space. Return the matrix.

⟨ Compute oriented border 25a ⟩ ≡

```
    def computeOrientedBordo3(nx, ny, nz):
        from larcc import signedCellularBoundary
        V, [VV, EV, FV, CV] = getBases(nx, ny, nz) ⟨ Get bases 23c ⟩
        boundaryMat = signedCellularBoundary(V, [VV, EV, FV, CV]) #lar-cc function
        return boundaryMat
    ◇
```

Macro referenced in 23a.

9 Laplacian smoothing

The idea is to smooth the model by successive refinement steps. Each step updates the coordinates of the vertices through the formula:

$$\bar{x}_i = \frac{1}{N}\sum_{j=1}^{N}\bar{x}_j \tag{1}$$

Where N is the number of adjacent vertices to node i, $\bar{x}_j$ is the position of the j-th adjacent vertex and $\bar{x}_i$ is the new position for node i. Here smoothing is done by two refinement steps. The track of the algorithm is to find the list of adjacent vertices for each vertex and apply a pyplasm function, *CCOMB*. It computes the convex combination of a list of vectors.

⟨ smooth 25b ⟩ ≡

```
    #two steps
    def makeSmoothing(V, FV):
        VV = adjVerts(V, FV) ⟨ adjacent vertices 26a ⟩
        V1 = AA(CCOMB)([[V[v] for v in adjs] for adjs in VV])
        V2 = AA(CCOMB)([[V1[v] for v in adjs] for adjs in VV])
        return V2 #new list of vertices
    ◇
```

Macro referenced in 8.

⟨ adjacent vertices 26a ⟩ ≡

```
def adjVerts(V, FV):
    n = len(V)
    VV = []
    V2V = adjacencyQuery(V, FV) ⟨ adjacency query 26b ⟩
    V2V = V2V.tocsr()
    for i in range(n):
        dataBuffer = V2V[i].tocoo().data
        colBuffer = V2V[i].tocoo().col
        row = []
        for val, j in zip(dataBuffer, colBuffer):
            if val == 2:
                row += [int(j)]
        VV += [row]
    return VV
```

◇

Macro referenced in 25b.

⟨ adjacency query 26b ⟩ ≡

```
def adjacencyQuery(V, FV):
    # dim = len(V[0])
    csrFV = csrCreate(FV)
    csrAdj = matrixProduct(csrTranspose(csrFV), csrFV)
    return csrAdj
```

◇

Macro referenced in 26a.

10 Generate a square mesh

The computed geometric model is a mesh of quads, saved in obj format. The aim of this function is to convert the model from binary to obj format.

⟨ Make obj 26c ⟩ ≡

```
def make_obj(nx, ny, nz, FILE_IN, OUT_DIR):

    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)
    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:
        readFile(V,FV,chunksize,FILE_IN,OUT_DIR) ⟨read bin file 27⟩
    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()')
◇
```

Macro referenced in 7.

10.1 Read a binary file

The conversion from binary to obj is done in two steps: convert the binary in two support files for vertices and faces and compute the vertices coordinates adding the segment offset.

⟨read bin file 27⟩ ≡

```
def readFile(V,FV,chunksize,inputFile,OUT_DIR):
    # outputVtx="outputVtx.obj",outputFaces="outputFaces.obj"):
    if not os.path.isfile(inputFile):
        print "File '%s' not found" % (inputFile)
        exit(-1)
    outputId = os.path.basename(inputFile).split('.')[0].split('-')[1]
    outputVtx=OUT_DIR+"/output-a-"+outputId+".stl"
    outputFaces=OUT_DIR+"/output-b-"+outputId+".stl"

    with open(inputFile, "rb") as file:
        with open(outputVtx, "w") as fileVertex:
            with open(outputFaces, "w") as fileFaces:

                vertex_count = 1
                old_vertex_count = vertex_count
                count = 0
```

```
       try:
           while True:
               count += 1

               zStart = struct.unpack('>I', file.read(4))[0]
               xStart = struct.unpack('>I', file.read(4))[0]
               yStart = struct.unpack('>I', file.read(4))[0]

               log(1, ["zStart, xStart, yStart = " + str(zStart) + "," + str(xStart) +
               #    zStart, xStart, yStart = LISTA_OFFSET[i].astype(float64)

               LISTA_VETTORI2 = np.zeros(chunksize,dtype=int32);

               # log(1, ["chunksize = " + str(chunksize)]);
               temp = file.read(chunksize);

               # log(1, ["chunksize = OK"]);

               i = 0
               timer_start("LISTA_VETTORI2 " + str(i));
               while (i < chunksize):
                   if (temp[i] == '\x01'):
                       LISTA_VETTORI2[i] = 1;
                   elif (temp[i] == '\xff'):
                       LISTA_VETTORI2[i] = -1;
                   i = i + 1;
               # TODO signum is wrong
               lista = LISTA_VETTORI2
               LISTA_VETTORI2 = np.abs(LISTA_VETTORI2)
               timer_stop();
               log(1, ["LISTA_VETTORI2[i] = " + str(i)]);

               timer_start("objectBoundaryChain ");
               l = len(LISTA_VETTORI2)
               objectBoundaryChain =
scipy.sparse.csr_matrix(LISTA_VETTORI2.reshape((l,1)))
               timer_stop();

               timer_start("csrChainToCellList " + str(i));
               b2cells = csrChainToCellList(objectBoundaryChain)

               timer_stop();

               timer_start("MKPOLS " + str(i));
               # orient FV
               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])

                        # import ipdb; ipdb.set_trace() #  noqa BREAKPOINT

                        # TODO old way is not efficient. It generates too much vertexes
                        vertex_count, old_vertex_count = writeToStlFilesOld(
                            fileVertex, fileFaces,
                            V, FVn,
                            xStart, yStart, zStart,
                            vertex_count, old_vertex_count,
                            b2cells
                        )
                        fileVertex.flush()
                        fileFaces.flush()

                    timer_stop();
                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) ])
```

◇

Macro referenced in 26c.

11 Visualization

Aim of this section is to visualize the obj model created through elaboration via LAR or pyplasm.

11.1 Pyplasm or LAR visualization

Visualization can use pyplasm visualizator or LAR, if explode functionality is wanted.

⟨ Visualize 29 ⟩ ≡

```
def visualize(V, FV, explode=False):
    if explode:
        visualize_lar(V, FV, explode) ⟨Visualize in LAR 30a⟩
    else:
        visualize_plasm(V, FV) ⟨Visualize in plasm 30b⟩
```

◇

Macro referenced in 4a.

⟨Visualize in LAR 30a⟩ ≡

```
def visualize_lar(V, FV, explode=False):
    import time
    t0 = time.time()
    mkpols = MKPOLS((V, FV))
    t1 = time.time()
    logger.debug("MKPOLS() done in %ss" % (str(t1 - t0)))
    if explode:
        VIEW(EXPLODE(1.2, 1.2, 1.2)(mkpols))
    else:
        struct = STRUCT(mkpols)
        t2 = time.time()
        logger.debug("STRUCT() done in %ss" % (str(t2 - t1)))
        VIEW(struct)
```

◇

Macro referenced in 29.

⟨Visualize in plasm 30b⟩ ≡

```
def visualize_plasm(V, FV):
    if len(FV[0]) > 3:
        FV = triangulateSquares(FV) ⟨Triangulation 30c⟩
    logger.debug("triangulation done")

    FV1 = (np.asarray(FV) + 1).tolist()
    logger.debug(" + 1 done")
    VIEW(MKPOL([V, FV1, []]))
```

◇

Macro referenced in 29.

11.2 Triangulation

Pyplasm needs quads to be triangulated.

⟨Triangulation 30c⟩ ≡

```
def triangulateSquares(F,
                       a=[0, 1, 2], b=[2, 3, 0],
                       c=[1, 0, 2], d=[3, 2, 0]
                       ):
    FT = []
    for face in F:
        FT.append([face[a[0]], face[a[1]], face[a[2]]])
        FT.append([face[b[0]], face[b[1]], face[b[2]]])
    return FT
◇
```

Macro referenced in 30b.

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