

ImagesToLARModel, a tool for creation of three-dimensional models from a stack of images

Danilo Salvati

October 27, 2015

Abstract

This is the abstract (we will use LAR [\[CL13\]](#))

Contents

1	Introduction	3
2	Exporting the library	3
2.1	Installing the library	31
3	Conclusions	31
3.1	Results	31
3.2	Further improvements	31
A	Utility functions	31
B	Tests	31

1 Introduction

2 Exporting the library

```
"src/ImagesToLARModel.jl" 3≡
module ImagesToLARModel
"""
Main module for the library. It starts conversion
taking configuration parameters
"""
require(string(Pkg.dir("ImagesToLARModel/src"), "/imagesConversion.jl"))

import JSON
import ImagesConversion

using Logging

export convertImagesToLARModel

function loadConfiguration(configurationFile)
"""
load parameters from JSON file

configurationFile: Path of the configuration file
"""

configuration = JSON.parse(configurationFile)

DEBUG_LEVELS = [DEBUG, INFO, WARNING, ERROR, CRITICAL]

try
    if configuration["parallelMerge"] == "true"
        parallelMerge = true
    else
        parallelMerge = false
    end
catch
    parallelMerge = false
end

return configuration["inputDirectory"], configuration["outputDirectory"], configuration["bestEffort"],
    configuration["nx"], configuration["ny"], configuration["nz"],
    DEBUG_LEVELS[configuration["DEBUG_LEVEL"]]

end
```

```

function convertImagesToLARModel(configurationFile)
    """
    Start conversion of a stack of images into a 3D model
    loading parameters from a JSON configuration file

    configurationFile: Path of the configuration file
    """
    inputDirectory, outputDirectory, bestImage, nx, ny, nz, DEBUG_LEVEL = loadConfiguration(open(
    convertImagesToLARModel(inputDirectory, outputDirectory, bestImage, nx, ny, nz, DEBUG_LEVEL)
end

function convertImagesToLARModel(inputDirectory, outputDirectory, bestImage,
                                nx, ny, nz, DEBUG_LEVEL = INFO, parallelMerge = false)
    """
    Start conversion of a stack of images into a 3D model

    inputDirectory: Directory containing the stack of images
    outputDirectory: Directory containing the output
    bestImage: Image chosen for centroids computation
    nx, ny, nz: Border dimensions (Possibly the biggest power of two of images dimensions)
    DEBUG_LEVEL: Debug level for Julia logger. It can be one of the following:
        - DEBUG
        - INFO
        - WARNING
        - ERROR
        - CRITICAL
    """
    # Create output directory
    try
        mkpath(outputDirectory)
    catch
    end

    Logging.configure(level=DEBUG_LEVEL)
    ImagesConversion.images2LARModel(nx, ny, nz, bestImage, inputDirectory, outputDirectory, par
end
end
◇

```

```

"src/imagesConversion.jl" 4≡
module ImagesConversion

require(string(Pkg.dir("ImagesToLARModel/src"), "/generateBorderMatrix.jl"))

```

```

require(string(Pkg.dir("ImagesToLARModel/src"), "/pngStack2Array3dJulia.jl"))
require(string(Pkg.dir("ImagesToLARModel/src"), "/lar2Julia.jl"))
require(string(Pkg.dir("ImagesToLARModel/src"), "/model2Obj.jl"))

import GenerateBorderMatrix
import PngStack2Array3dJulia
import Lar2Julia
import Model2Obj

import JSON

using PyCall
@pyimport scipy.sparse as Pysparse

using Logging

export images2LARModel

"""
This is main module for converting a stack
of images into a 3d model
"""

function images2LARModel(nx, ny, nz, bestImage, inputDirectory, outputDirectory, parallelMerge)
    """
    Convert a stack of images into a 3d model
    """

    info("Starting model creation")

    numberOfClusters = 2 # Number of clusters for
                        # images segmentation

    info("Moving images into temp directory")
    try
        mkdir(string(outputDirectory, "TEMP"))
    catch
    end

    tempDirectory = string(outputDirectory, "TEMP/")

    newBestImage = PngStack2Array3dJulia.convertImages(inputDirectory, tempDirectory, bestImage)

    imageWidth, imageHeight = PngStack2Array3dJulia.getImageData(string(tempDirectory, newBestImage))
    imageDepth = length(readdir(tempDirectory))

```

```

# Computing border matrix
info("Computing border matrix")
try
    mkdir(string(outputDirectory, "BORDERS"))
catch
end
borderFilename = GenerateBorderMatrix.getOriented3BorderPath(string(outputDirectory, "BORDERS")

# Starting images conversion and border computation
info("Starting images conversion")
startImageConversion(tempDirectory, newBestImage, outputDirectory, borderFilename,
                    imageHeight, imageWidth, imageDepth,
                    nx, ny, nz,
                    numberOfClusters, parallelMerge)

end

function startImageConversion(sliceDirectory, bestImage, outputDirectory, borderFilename,
                            imageHeight, imageWidth, imageDepth,
                            imageDx, imageDy, imageDz,
                            numberOfClusters, parallelMerge)

    """
    Support function for converting a stack of images into a model

    sliceDirectory: directory containing the image stack
    imageForCentroids: image chosen for centroid computation
    """

    # Create clusters for image segmentation
    info("Computing image centroids")
    debug("Best image = ", bestImage)
    centroidsCalc = PngStack2Array3dJulia.calculateClusterCentroids(sliceDirectory, bestImage, n
    debug(string("centroids = ", centroidsCalc))

    try
        mkdir(string(outputDirectory, "BORDERS"))
    catch
    end
    debug(string("Opening border file: ", "border_", imageDx, "-", imageDy, "-", imageDz, ".json"))
    boundaryMat = getBorderMatrix(string(outputDirectory, "BORDERS/", "border_", imageDx, "-",
                                    imageDy, "-", imageDz, ".json"))

    beginImageStack = 0
    endImage = beginImageStack

```

```

info("Converting images into a 3d model")
tasks = Array{RemoteRef, 0}()
for zBlock in 0:(imageDepth / imageDz - 1)
    startImage = endImage
    endImage = startImage + imageDz
    info("StartImage = ", startImage)
    info("endImage = ", endImage)

    task = @spawn imageConversionProcess(sliceDirectory, outputDirectory,
                                        beginImageStack, startImage, endImage,
                                        imageDx, imageDy, imageDz,
                                        imageHeight, imageWidth,
                                        centroidsCalc, boundaryMat)

    push!(tasks, task)

end

# Waiting for processes completion
for task in tasks
    wait(task)
end

info("Merging obj models")
if parallelMerge
    Model2Obj.mergeObjParallel(string(outputDirectory, "MODELS"))
else
    Model2Obj.mergeObj(string(outputDirectory, "MODELS"))
end

end

function imageConversionProcess(sliceDirectory, outputDirectory,
                               beginImageStack, startImage, endImage,
                               imageDx, imageDy, imageDz,
                               imageHeight, imageWidth,
                               centroids, boundaryMat)

    """
    Support function for converting a stack of image on a single
    independent process
    """

    info("Transforming png data into 3d array")
    theImage = PngStack2Array3dJulia.pngstack2array3d(sliceDirectory, startImage, endImage, centroids)

    centroidsSorted = sort(vec(reshape(centroids, 1, 2)))
    foreground = centroidsSorted[2]

```

```

background = centroidsSorted[1]
debug(string("background = ", background, " foreground = ", foreground))

# V and FV contains vertices and faces of this part of model
V = Array(Array{Int}, 0)
FV = Array(Array{Int}, 0)
facesOffset = 0
for xBlock in 0:(imageHeight / imageDx - 1)
    for yBlock in 0:(imageWidth / imageDy - 1)
        yStart = xBlock * imageDx
        xStart = yBlock * imageDy
        #xEnd = xStart + imageDx
        #yEnd = yStart + imageDy
        xEnd = xStart + imageDy
        yEnd = yStart + imageDx
        debug("*****")
        debug(string("xStart = ", xStart, " xEnd = ", xEnd))
        debug(string("yStart = ", yStart, " yEnd = ", yEnd))
        debug("theImage dimensions: ", size(theImage)[1], " ", size(theImage[1])[1], " ", size(t

# Getting a slice of theImage array

image = Array{UInt8, (convert{Int, length(theImage)}, convert{Int, xEnd - xStart}, convert{Int, yEnd - yStart})}
debug("image size: ", size(image))
for z in 1:length(theImage)
    for x in 1 : (xEnd - xStart)
        for y in 1 : (yEnd - yStart)
            image[z, x, y] = theImage[z][x + xStart, y + yStart]
        end
    end
end

nx, ny, nz = size(image)
chains3D = Array{UInt8, 0}
zStart = startImage - beginImageStack
for y in 0:(nx - 1)
    for x in 0:(ny - 1)
        for z in 0:(nz - 1)
            if(image[z + 1, x + 1, y + 1] == foreground)
                push!(chains3D, y + ny * (x + nx * z))
            end
        end
    end
end

if(length(chains3D) != 0)

```



```

        # Computing boundary chain
        debug("chains3d = ", chains3D)
        debug("Computing boundary chain")
        objectBoundaryChain = Lar2Julia.larBoundaryChain(boundaryMat, chains3D)
        debug("Converting models into obj")
        try
            mkdir(string(outputDirectory, "MODELS"))
        catch
        end
        # IMPORTANT: inverting xStart and yStart for obtaining correct rotation of the model
        V_part, FV_part = Model2Obj.computeModel(imageDx, imageDy, imageDz, yStart, xStart, zStart)
        facesOffset += length(V_part)
        append!(V, V_part)
        append!(FV, FV_part)
    else
        debug("Model is empty")
    end
end
end
outputFilename = string(outputDirectory, "MODELS/model_output_", startImage, "_", endImage)
Model2Obj.writeToObj(V, FV, outputFilename)
end

function getBorderMatrix(borderFilename)
    """
    TO REMOVE WHEN PORTING OF LARCC IN JULIA IS COMPLETED

    Get the border matrix from json file and convert it in
    CSC format
    """
    # Loading borderMatrix from json file
    borderData = JSON.parsefile(borderFilename)
    row = Array{Int64, 1}(length(borderData["ROW"]))
    col = Array{Int64, 1}(length(borderData["COL"]))
    data = Array{Int64, 1}(length(borderData["DATA"]))

    for i in 1: length(borderData["ROW"])
        row[i] = borderData["ROW"][i]
    end

    for i in 1: length(borderData["COL"])
        col[i] = borderData["COL"][i]
    end

    for i in 1: length(borderData["DATA"])
        data[i] = borderData["DATA"][i]
    end
end

```

```

    end

    # Converting csr matrix to csc
    csrBorderMatrix = Pysparse.csr_matrix((data,col,row), shape=(borderData["ROWCOUNT"],borderDa
    denseMatrix = pycall(csrBorderMatrix["toarray"],PyAny)

    cscBoundaryMat = sparse(denseMatrix)

    return cscBoundaryMat

end
end
◇

"src/generateBorderMatrix.jl" 10≡
module GenerateBorderMatrix
"""
Module for generation of the boundary matrix
"""

type MatrixObject
    ROWCOUNT
    COLCOUNT
    ROW
    COL
    DATA
end

export computeOriented3Border, writeBorder, getOriented3BorderPath

require(string(Pkg.dir("ImagesToLARModel/src"), "/larUtils.jl"))

import LARUtils
using PyCall

import JSON

@pyimport sys
unshift!(PyVector(pyimport("sys")["path"]), "") # Search for python modules in folder
# Search for python modules in package folder
unshift!(PyVector(pyimport("sys")["path"]), Pkg.dir("ImagesToLARModel/src"))
@pyimport larcc # Importing larcc from local folder

```

```

# Compute the 3-border operator
function computeOriented3Border(nx, ny, nz)
    """
    Compute the 3-border matrix using a modified
    version of larcc
    """
    V, bases = LARUtils.getBases(nx, ny, nz)
    boundaryMat = larcc.signedCellularBoundary(V, bases)
    return boundaryMat
end

function writeBorder(boundaryMatrix, outputFile)
    """
    Write 3-border matrix on json file

    boundaryMatrix: matrix to write on file
    outputFile: path of the outputFile
    """

    rowcount = boundaryMatrix[:shape][1]
    colcount = boundaryMatrix[:shape][2]

    row = boundaryMatrix[:indptr]
    col = boundaryMatrix[:indices]
    data = boundaryMatrix[:data]

    # Writing informations on file
    outfile = open(outputFile, "w")

    matrixObj = MatrixObject(rowcount, colcount, row, col, data)
    JSON.print(outfile, matrixObj)
    close(outfile)
end

function getOriented3BorderPath(borderPath, nx, ny, nz)
    """
    Try reading 3-border matrix from file. If it fails matrix
    is computed and saved on disk in JSON format

    borderPath: path of border directory
    nx, ny, nz: image dimensions
    """

```

```

    filename = string(borderPath, "/border_", nx, "-", ny, "-", nz, ".json")
    if !isfile(filename)
        border = computeOriented3Border(nx, ny, nz)
        writeBorder(border, filename)
    end
    return filename

end
end
◇

```

```

"src/lar2Julia.jl" 12≡
module Lar2Julia
"""
    larcc functions for Julia
"""
export larBoundaryChain, cscChainToCellList

import JSON

using Logging

function larBoundaryChain(cscBoundaryMat, brcCellList)
    """
        Compute boundary chains
    """

    # Computing boundary chains
    n = size(cscBoundaryMat)[1]
    m = size(cscBoundaryMat)[2]

    debug("Boundary matrix size: ", n, "\t", m)

    data = ones{Int64, length(brcCellList)}

    i = Array{Int64, length(brcCellList)}
    for k in 1:length(brcCellList)
        i[k] = brcCellList[k] + 1
    end

    j = ones{Int64, length(brcCellList)}

    debug("cscChain rows length: ", length(i))

```

```

debug("cscChain columns length: ", length(j))
debug("cscChain data length: ", length(brcCellList))

debug("rows ", i)
debug("columns ", j)
debug("data ", data)

cscChain = sparse(i, j, data, m, 1)
cscmat = cscBoundaryMat * cscChain
out = cscBinFilter(cscmat)
return out
end

function cscBinFilter(CSCm)
    k = 1
    data = nonzeros(CSCm)
    sgArray = copysign(1, data)

    while k <= nnz(CSCm)
        if data[k] % 2 == 1 || data[k] % 2 == -1
            data[k] = 1 * sgArray[k]
        else
            data[k] = 0
        end
        k += 1
    end

    return CSCm
end

function cscChainToCellList(CSCm)
    """
    Get a csc containing a chain and returns
    the cell list of the "+1" oriented faces
    """
    data = nonzeros(CSCm)
    # Now I need to remove zero element (problem with Julia nonzeros)
    nonzeroData = Array{Int64, 0}
    for n in data
        if n != 0
            push!(nonzeroData, n)
        end
    end

    cellList = Array{Int64, 0}
    for (k, theRow) in enumerate(findn(CSCm)[1])

```

```

        if nonzeroData[k] == 1
            push!(cellList, theRow)
        end
    end
    return cellList
end
end
◇

```

```

"src/larUtils.jl" 14≡
module LARUtils
"""
Utility functions for extracting 3d models from images
"""
export ind, invertIndex, getBases

function ind(x, y, z, nx, ny)
    """
    Transform coordinates into linearized matrix indexes
    """
    return x + (nx+1) * (y + (ny+1) * (z))
end

function invertIndex(nx,ny,nz)
    """
    Invert indexes
    """
    nx, ny, nz = nx + 1, ny + 1, nz + 1
    function invertIndex0(offset)
        a0, b0 = trunc(offset / nx), offset % nx
        a1, b1 = trunc(a0 / ny), a0 % ny
        a2, b2 = trunc(a1 / nz), a1 % nz
        return b0, b1, b2
    end
    return invertIndex0
end

function getBases(nx, ny, nz)
    """
    Compute all LAR relations
    """

```

```

function the3Dcell(coords)
  x,y,z = coords
  return [ind(x,y,z,nx,ny),ind(x+1,y,z,nx,ny),ind(x,y+1,z,nx,ny),ind(x,y,z+1,nx,ny),ind(x+1,y,z+1,nx,ny),ind(x,y+1,z+1,nx,ny),ind(x+1,y+1,z+1,nx,ny)]
end

# Calculating vertex coordinates (nx * ny * nz)
V = Array{Int64}[]
for z in 0:nz
  for y in 0:ny
    for x in 0:nx
      push!(V,[x,y,z])
    end
  end
end

# Building CV relationship
CV = Array{Int64}[]
for z in 0:nz-1
  for y in 0:ny-1
    for x in 0:nx-1
      push!(CV,the3Dcell([x,y,z]))
    end
  end
end

# Building FV relationship
FV = Array{Int64}[]
v2coords = invertIndex(nx,ny,nz)

for h in 0:(length(V)-1)
  x,y,z = v2coords(h)

  if (x < nx) && (y < ny)
    push!(FV, [h,ind(x+1,y,z,nx,ny),ind(x,y+1,z,nx,ny),ind(x+1,y+1,z,nx,ny)])
  end

  if (x < nx) && (z < nz)
    push!(FV, [h,ind(x+1,y,z,nx,ny),ind(x,y,z+1,nx,ny),ind(x+1,y,z+1,nx,ny)])
  end

  if (y < ny) && (z < nz)
    push!(FV, [h,ind(x,y+1,z,nx,ny),ind(x,y,z+1,nx,ny),ind(x,y+1,z+1,nx,ny)])
  end
end

```

```

end

# Building VV relationship
VV = map((x)->[x], 0:length(V)-1)

# Building EV relationship
EV = Array{Int64}[]
for h in 0:length(V)-1
    x,y,z = v2coords(h)
    if (x < nx)
        push!(EV, [h,ind(x+1,y,z,nx,ny)])
    end
    if (y < ny)
        push!(EV, [h,ind(x,y+1,z,nx,ny)])
    end
    if (z < nz)
        push!(EV, [h,ind(x,y,z+1,nx,ny)])
    end
end
end

# return all basis
return V, (VV, EV, FV, CV)
end
end
◇

```

```

"src/model2Obj.jl" 16≡
module Model2Obj
"""
Module that takes a 3d model and write it on
obj files
"""

require(string(Pkg.dir("ImagesToLARModel/src"), "/larUtils.jl"))

import LARUtils

using Logging

export writeToObj, mergeObj, computeModel, mergeObjParallel

```



```

function lessThanVertices(v1, v2)
    """
    Utility function for comparing vertices coordinates
    """

    if v1[1] == v2[2]
        if v1[2] == v2[2]
            return v1[3] < v2[3]
        end
        return v1[2] < v2[2]
    end
    return v1[1] < v2[2]
end

function removeDoubleVerticesAndFaces(V, FV, facesOffset)
    """
    Removes double vertices and faces from a LAR model

    V: Array containing all vertices
    FV: Array containing all faces
    facesOffset: offset for faces indices
    """

    newV, indices = removeDoubleVertices(V)
    reindexedFaces = reindexVerticesInFaces(FV, indices, facesOffset)
    newFV = unique(FV)

    return newV, newFV
end

function removeDoubleVertices(V)
    """
    Remove double vertices from a LAR model

    V: Array containing all vertices of the model
    """

    # Sort the vertices list and returns the ordered indices
    orderedIndices = sortperm(V, lt = lessThanVertices, alg=MergeSort)

    orderedVerticesAndIndices = collect(zip(sort(V, lt = lessThanVertices),
                                             orderedIndices))

    newVertices = Array{Array{Int}, 0}()
    indices = zeros{Int, length(V)}
    prevv = Nothing

```

```

i = 1
for (v, ind) in orderedVerticesAndIndices
    if v == prevv
        indices[ind] = i - 1
    else
        push!(newVertices, v)
        indices[ind] = i
        i += 1
        prevv = v
    end
end
return newVertices, indices
end

function reindexVerticesInFaces(FV, indices, offset)
    """
    Reindex vertices indexes in faces array

    FV: Faces array of the LAR model
    indices: new Indices for faces
    offset: offset for faces indices
    """

    for f in FV
        for i in 1: length(f)
            f[i] = indices[f[i] - offset] + offset
        end
    end
    return FV
end

function computeModel(imageDx, imageDy, imageDz,
                      xStart, yStart, zStart,
                      facesOffset, objectBoundaryChain)
    """
    Takes the boundary chain of a part of the entire model
    and returns a LAR model

    imageDx, imageDy, imageDz: Boundary dimensions
    xStart, yStart, zStart: Offset of this part of the model
    facesOffset: Offset for the faces
    objectBoundaryChain: Sparse csc matrix containing the cells
    """

    V, bases = LARUtils.getBases(imageDx, imageDy, imageDz)
    FV = bases[3]

```

```

V_model = Array{Array{Int}, 0}
FV_model = Array{Array{Int}, 0}

vertex_count = 1

#b2cells = Lar2Julia.cscChainToCellList(objectBoundaryChain)
# Get all cells (independently from orientation)
b2cells = findn(objectBoundaryChain)[1]

debug("b2cells = ", b2cells)

for f in b2cells
    old_vertex_count = vertex_count
    for vtx in FV[f]
        push!(V_model, [convert{Int}(V[vtx + 1][1] + xStart),
                        convert{Int64}(V[vtx + 1][2] + yStart),
                        convert{Int64}(V[vtx + 1][3] + zStart)])
        vertex_count += 1
    end

    push!(FV_model, [old_vertex_count + facesOffset, old_vertex_count + 1 + facesOffset, old_v
    push!(FV_model, [old_vertex_count + facesOffset, old_vertex_count + 3 + facesOffset, old_v
end

# Removing double vertices
return removeDoubleVerticesAndFaces(V_model, FV_model, facesOffset)

end

function writeToObj(V, FV, outputFilename)
    """
    Take a LAR model and write it on obj file

    V: array containing vertices coordinates
    FV: array containing faces
    outputFilename: prefix for the output files
    """

    outputVtx = string(outputFilename, "_vtx.stl")
    outputFaces = string(outputFilename, "_faces.stl")

    fileVertex = open(outputVtx, "w")
    fileFaces = open(outputFaces, "w")

    for v in V

```

```

        write(fileVertex, "v ")
        write(fileVertex, string(v[1], " "))
        write(fileVertex, string(v[2], " "))
        write(fileVertex, string(v[3], "\n"))
    end

    for f in FV

        write(fileFaces, "f ")
        write(fileFaces, string(f[1], " "))
        write(fileFaces, string(f[2], " "))
        write(fileFaces, string(f[3], "\n"))
    end

    close(fileVertex)
    close(fileFaces)

end

function mergeObj(modelDirectory)
    """
    Merge stl files in a single obj file

    modelDirectory: directory containing models
    """

    files = readdir(modelDirectory)
    vertices_files = files[find(s -> contains(s,string("_vtx.stl")), files)]
    faces_files = files[find(s -> contains(s,string("_faces.stl")), files)]
    obj_file = open(string(modelDirectory, "/", "model.obj"), "w") # Output file

    vertices_counts = Array{Int64, length(vertices_files)}
    number_of_vertices = 0
    for i in 1:length(vertices_files)
        vtx_file = vertices_files[i]
        f = open(string(modelDirectory, "/", vtx_file))
        debug("Opening ", vtx_file)

        # Writing vertices on the obj file
        for ln in eachline(f)
            write(obj_file, ln)
            number_of_vertices += 1
        end
        # Saving number of vertices
        vertices_counts[i] = number_of_vertices
        close(f)
    end
end

```

```

end

for i in 1 : length(faces_files)
    faces_file = faces_files[i]
    f = open(string(modelDirectory, "/", faces_file))
    debug("Opening ", faces_file)
    for ln in eachline(f)
        splitted = split(ln)
        write(obj_file, "f ")
        if i > 1
            write(obj_file, string(parse(splitted[2]) + vertices_counts[i - 1], " "))
            write(obj_file, string(parse(splitted[3]) + vertices_counts[i - 1], " "))
            write(obj_file, string(parse(splitted[4]) + vertices_counts[i - 1]))
        else
            write(obj_file, string(splitted[2], " "))
            write(obj_file, string(splitted[3], " "))
            write(obj_file, splitted[4])
        end
        write(obj_file, "\n")
    end
    close(f)
end
close(obj_file)

# Removing all tmp files
for vtx_file in vertices_files
    rm(string(modelDirectory, "/", vtx_file))
end

for fcs_file in faces_files
    rm(string(modelDirectory, "/", fcs_file))
end

end

function assignTasks(startInd, endInd, taskArray)
    """
    This function choose the first files to merge
    creating a tree where number of processes is maximized

    startInd: starting index for array subdivision
    endInd: end index for array subdivision
    taskArray: array containing indices of files to merge for first
    """
    if (endInd - startInd == 2)
        push!(taskArray, startInd)
    end
end

```

```

elseif (endInd - startInd < 2)
    if (endInd % 4 != 0 && startInd != endInd)
        # Stop recursion on this branch
        push!(taskArray, startInd)
    end
    # Stop recursion doing nothing
else
    assignTasks(startInd, startInd + trunc((endInd - startInd) / 2), taskArray)
    assignTasks(startInd + trunc((endInd - startInd) / 2) + 1, endInd, taskArray)
end
end

function mergeVerticesFiles(file1, file2, startOffset)
    """
    Support function for merging two vertices files.
    Returns the number of vertices of the merged file

    file1: path of the first file
    file2: path of the second file
    startOffset: starting face offset for second file
    """

    f1 = open(file1, "a")

    f2 = open(file2)
    debug("Merging ", file2)
    number_of_vertices = startOffset
    for ln in eachline(f2)
        write(f1, ln)
        number_of_vertices += 1
    end
    close(f2)

    close(f1)

    return number_of_vertices
end

function mergeFacesFiles(file1, file2, facesOffset)
    """
    Support function for merging two faces files

    file1: path of the first file
    file2: path of the second file
    facesOffset: offset for faces
    """

```

```

"""

f1 = open(file1, "a")

f2 = open(file2)
for ln in eachline(f2)
    splitted = split(ln)
    write(f1, "f ")
    write(f1, string(parse(splitted[2]) + facesOffset, " "))
    write(f1, string(parse(splitted[3]) + facesOffset, " "))
    write(f1, string(parse(splitted[4]) + facesOffset, "\n"))
end
close(f2)

close(f1)
end

function mergeObjProcesses(fileArray, facesOffset = Nothing)
"""
Merge files on a single process

fileArray: Array containing files that will be merged
facesOffset (optional): if merging faces files, this array contains
    offsets for every file
"""

if(contains(fileArray[1], string("_vtx.stl")))
    # Merging vertices files
    offsets = Array{Int, 0}
    push!(offsets, countlines(fileArray[1]))
    vertices_count = mergeVerticesFiles(fileArray[1], fileArray[2], countlines(fileArray[1]))
    rm(fileArray[2]) # Removing merged file
    push!(offsets, vertices_count)
    for i in 3: length(fileArray)
        vertices_count = mergeVerticesFiles(fileArray[1], fileArray[i], vertices_count)
        rm(fileArray[i]) # Removing merged file
        push!(offsets, vertices_count)
    end
    return offsets
else
    # Merging faces files
    mergeFacesFiles(fileArray[1], fileArray[2], facesOffset[1])
    rm(fileArray[2]) # Removing merged file
    for i in 3 : length(fileArray)
        mergeFacesFiles(fileArray[1], fileArray[i], facesOffset[i - 1])
        rm(fileArray[i]) # Removing merged file
    end
end
end

```

```

        end
    end
end

function mergeObjHelper(vertices_files, faces_files)
    """
    Support function for mergeObj. It takes vertices and faces files
    and execute a single merging step

    vertices_files: Array containing vertices files
    faces_files: Array containing faces files
    """

    numberOfImages = length(vertices_files)
    taskArray = Array{Int, 0}
    assignTasks(1, numberOfImages, taskArray)

    # Now taskArray contains first files to merge
    numberOfVertices = Array{Int, 0}
    tasks = Array{RemoteRef, 0}
    for i in 1 : length(taskArray) - 1
        task = @spawn mergeObjProcesses(vertices_files[taskArray[i] : (taskArray[i + 1] - 1)])
        push!(tasks, task)
        #append!(numberOfVertices, mergeObjProcesses(vertices_files[taskArray[i] : (taskArray[i + 1] - 1)]))
    end

    # Merging last vertices files
    task = @spawn mergeObjProcesses(vertices_files[taskArray[length(taskArray)] : end])
    push!(tasks, task)
    #append!(numberOfVertices, mergeObjProcesses(vertices_files[taskArray[length(taskArray)] : end]))

    for task in tasks
        append!(numberOfVertices, fetch(task))
    end

    debug("NumberOfVertices = ", numberOfVertices)

    # Merging faces files
    tasks = Array{RemoteRef, 0}
    for i in 1 : length(taskArray) - 1

        task = @spawn mergeObjProcesses(faces_files[taskArray[i] : (taskArray[i + 1] - 1)],
                                       numberOfVertices[taskArray[i] : (taskArray[i + 1] - 1)])
        push!(tasks, task)

        #mergeObjProcesses(faces_files[taskArray[i] : (taskArray[i + 1] - 1)],

```



```

        #                                numberOfVertices[taskArray[i] : (taskArray[i + 1] - 1)])
    end

    #Merging last faces files
    task = @spawn mergeObjProcesses(faces_files[taskArray[length(taskArray)] : end],
                                    numberOfVertices[taskArray[length(taskArray)] : end])

    push!(tasks, task)
    #mergeObjProcesses(faces_files[taskArray[length(taskArray)] : end],
    #                   numberOfVertices[taskArray[length(taskArray)] : end])

    for task in tasks
        wait(task)
    end

end

function mergeObjParallel(modelDirectory)
    """
    Merge stl files in a single obj file using a parallel
    approach. Files will be recursively merged two by two
    generating a tree where number of processes for every
    step is maximized
    Actually use of this function is discouraged. In fact
    speedup is influenced by disk speed. It could work on
    particular systems with parallel accesses on disks

    modelDirectory: directory containing models
    """

    files = readdir(modelDirectory)

    # Appending directory path to every file
    files = map((s) -> string(modelDirectory, "/", s), files)

    # While we have more than one vtx file and one faces file
    while(length(files) != 2)
        vertices_files = files[find(s -> contains(s,string("_vtx.stl")), files)]
        faces_files = files[find(s -> contains(s,string("_faces.stl")), files)]

        # Merging files
        mergeObjHelper(vertices_files, faces_files)

        files = readdir(modelDirectory)
        files = map((s) -> string(modelDirectory, "/", s), files)
    end
end

```

```

mergeVerticesFiles(files[2], files[1], 0)
mv(files[2], string(modelDirectory, "/model.obj"))
rm(files[1])

end
end
◇

"src/pngStack2Array3dJulia.jl" 26≡
module PngStack2Array3dJulia

"""
This module loads a stack of png files returning
an array of pixel values divided into segments
"""

export calculateClusterCentroids, pngstack2array3d, getImageData, convertImages

using Images # For loading png images
using Colors # For grayscale images
using PyCall # For including python clustering
using Logging
@pyimport scipy.ndimage as ndimage
@pyimport scipy.cluster.vq as cluster

NOISE_SHAPE_DETECT=10

function getImageData(imageFile)
"""
Get width and height from a png image
"""

input = open(imageFile, "r")
data = readbytes(input, 24)

if (data[2:4] != [80, 78, 71] && data[13:16] != [73, 72, 68, 82])
    error("This is not a png image")
end

w = data[17:20]
h = data[21:24]

```

```

width = reinterpret{Int32, reverse(w)}[1]
height = reinterpret{Int32, reverse(h)}[1]

close(input)

return width, height
end

function calculateClusterCentroids(path, image, numberOfClusters = 2)
    """
    Loads an image and calculate cluster centroids for segmentation

    path: Path of the image folder
    image: name of the image
    numberOfClusters: number of desired clusters
    """
    imageFilename = string(path, image)

    img = imread(imageFilename) # Open png image with Julia Package

    rgb_img = convert{Image{ColorTypes.RGB}}, img)
    gray_img = convert{Image{ColorTypes.Gray}}, rgb_img)
    imArray = raw(gray_img)

    imageWidth = size(imArray)[1]
    imageHeight = size(imArray)[2]

    # Getting pixel values and saving them with another shape
    image3d = Array{Array{UInt8,2}, 0}

    # Inserting page on another list and reshaping
    push!(image3d, imArray)
    pixel = reshape(image3d[1], (imageWidth * imageHeight), 1)

    # Segmenting image using kmeans
    # https://en.wikipedia.org/wiki/Image\_segmentation#Clustering\_methods

    centroids,_ = cluster.kmeans(pixel, numberOfClusters)

    return centroids
end

function pngstack2array3d(path, minSlice, maxSlice, centroids)
    """

```

```

Import a stack of PNG images into a 3d array

path: path of images directory
minSlice and maxSlice: number of first and last slice
centroids: centroids for image segmentation
"""

# image3d contains all images values
image3d = Array{Array{UInt8,2}, 0}

debug("maxSlice = ", maxSlice, " minSlice = ", minSlice)
files = readdir(path)

for slice in minSlice : (maxSlice - 1)
    debug("slice = ", slice)
    imageFilename = string(path, files[slice + 1])
    debug("image name: ", imageFilename)
    img = imread(imageFilename) # Open png image with Julia Package

    # Converting image in grayscale
    rgb_img = convert{ColorTypes.RGB}, img)
    gray_img = convert{ColorTypes.Gray}, rgb_img)
    imArray = raw(gray_img) # Putting pixel values into RAW 3d array
    debug("imArray size: ", size(imArray))

    # Inserting page on another list and reshaping
    push!(image3d, imArray)

end

# Removing noise using a median filter and quantization
for page in 1:length(image3d)

    # Denoising
    image3d[page] = ndimage.median_filter(image3d[page], NOISE_SHAPE_DETECT)

    # Image Quantization
    debug("page = ", page)
    debug("image3d[page] dimensions: ", size(image3d[page])[1], "\t", size(image3d[page])[2])
    pixel = reshape(image3d[page], size(image3d[page])[1] * size(image3d[page])[2] , 1)
    qnt,_ = cluster.vq(pixel,centroids)

    # Reshaping quantization result
    centers_idx = reshape(qnt, size(image3d[page],1), size(image3d[page],2))
    #centers_idx = reshape(qnt, size(image3d[page]))

```

```

    # Inserting quantized values into 3d image array
    tmp = Array(_UINT8, size(image3d[page],1), size(image3d[page],2))

    for j in 1:size(image3d[1],2)
        for i in 1:size(image3d[1],1)
            tmp[i,j] = centroids[centers_idx[i,j] + 1]
        end
    end

    image3d[page] = tmp

end

return image3d
end

function convertImages(inputPath, outputPath, bestImage)
    """
    Get all images contained in inputPath directory
    saving them in outputPath directory in png format.
    If images have one of two odd dimensions, they will be resized
    and if folder contains an odd number of images another one will be
    added

    inputPath: Directory containing input images
    outputPath: Temporary directory containing png images
    bestImage: Image chosen for centroids computation

    Returns the new name for the best image
    """

    imageFiles = readdir(inputPath)
    numberOfImages = length(imageFiles)
    outputPrefix = ""
    for i in 1: length(string(numberOfImages)) - 1
        outputPrefix = string(outputPrefix,"0")
    end

    newBestImage = ""
    imageNumber = 0
    for imageFile in imageFiles
        img = imread(string(inputPath, imageFile))

        # resizing images if they do not have even dimensions
        dim = size(img)
        if(dim[1] % 2 != 0)

```

```

        debug("Image has odd x; resizing")
        xrange = 1: dim[1] - 1
    else
        xrange = 1: dim[1]
    end

    if(dim[2] % 2 != 0)
        debug("Image has odd y; resizing")
        yrange = 1: dim[2] - 1
    else
        yrange = 1: dim[2]
    end

    img = subim(img, xrange, yrange)

    outputFilename = string(outputPath, outputPrefix[length(string(imageNumber)):end], imageNumber)
    imwrite(img, outputFilename)

    # Searching the best image
    if(imageFile == bestImage)
        newBestImage = string(outputPrefix[length(string(imageNumber)):end], imageNumber, ".png")
    end

    imageNumber += 1
end

# Adding another image if they are odd
if(numberOfImages % 2 != 0)
    debug("Odd images, adding one")
    bestImage = imread(string(outputPath, "/", newBestImage))
    imArray = zeros(Uint8, size(bestImage))
    img = grayim(imArray)
    outputFilename = string(outputPath, "/", outputPrefix[length(string(imageNumber)):end], imageNumber)
    imwrite(img, outputFilename)
end

return newBestImage
end

end
◇

```

3 Conclusions

3.2 Further improvements

[CL13] CVD-Lab, *Linear algebraic representation*, Tech. Report 13-00, Roma Tre University, October 2013.

B Tests

```
"test/generateBorderMatrix.jl" 31≡  
    push!(LOAD_PATH, "../..")  
    import GenerateBorderMatrix  
    import JSON  
    using Base.Test  
  
function testComputeOriented3Border()  
    """  
        Test function for computeOriented3Border  
    """  
    boundaryMatrix = GenerateBorderMatrix.computeOriented3Border(2,2,2)  
  
    rowcount = boundaryMatrix[:shape][1]  
    @test rowcount == 36  
    colcount = boundaryMatrix[:shape][2]  
    @test colcount == 8  
    row = boundaryMatrix[:,indptr]  
    @test row == [0,1,2,3,4,5,7,8,9,11,12,13,15,17,18,19,20,22,23,24,26,27,29,30,32,34,35,37,39,  
    col = boundaryMatrix[:,indices]  
    @test col == [0,0,0,1,1,0,1,1,2,0,2,2,3,1,3,2,3,3,2,3,0,4,4,4,1,5,5,4,5,5,2,6,4,6,6,3,7,5,7,  
    data = boundaryMatrix[:,data]  
    @test data == [-1,1,-1,-1,1,1,-1,1,-1,-1,1,-1,-1,-1,1,-1,1,-1,-1,1,-1,1,-1,-1,-1,1,-1,1,-1,1,-1,1,1,-1,1,1,  
  
end  
  
function testWriteBorder()
```



```

    """

    width, height = PngStack2Array3dJulia.getImageData("images/0.png")

    @test width == 50
    @test height == 50

end

function testCalculateClusterCentroids()
    """
    Test function for calculateClusterCentroids
    """
    path = "images/"
    image = 0
    centroids = PngStack2Array3dJulia.calculateClusterCentroids(path, image, 2)

    expected = [0, 253]
    centroids = vec(reshape(centroids, 1, 2))

    @test sort(centroids) == expected
end

function testPngstack2array3d()
    """
    Test function for pngstack2array3d
    """
    path = "images/"
    minSlice = 0
    maxSlice = 4
    centroids = PngStack2Array3dJulia.calculateClusterCentroids(path, 0, 2)
    image3d = PngStack2Array3dJulia.pngstack2array3d(path, minSlice, maxSlice, centroids)

    @test size(image3d)[1] == 5
    @test size(image3d[1])[1] == 50
    @test size(image3d[1])[2] == 200

end

function executeAllTests()
    @time testCalculateClusterCentroids()
    @time testPngstack2array3d()
    @time testGetImageData()
    println("Tests completed.")
end

```

```
executeAllTests()
```

◇

Test for LAR utilities

```
"test/LARUtils.jl" 34≡
push!(LOAD_PATH, "../..")
import LARUtils
using Base.Test

function testInd()
    """
    Test function for ind
    """

    nx = 2
    ny = 2

    @test LARUtils.ind(0, 0, 0, nx, ny) == 0
    @test LARUtils.ind(1, 1, 1, nx, ny) == 13
    @test LARUtils.ind(2, 5, 4, nx, ny) == 53
    @test LARUtils.ind(1, 1, 1, nx, ny) == 13
    @test LARUtils.ind(2, 7, 1, nx, ny) == 32
    @test LARUtils.ind(1, 0, 3, nx, ny) == 28
end

function executeAllTests()
    @time testInd()
    println("Tests completed.")
end

executeAllTests()

◇
```