3D Region Growing Implementation

University of Trento
Signal, Image and Video
Professors:Francesco De Natale and Andrea Rosani
Date: 22-06-2021

Gabriele Berrera[1], Emma Busarello[1] and Francesco Penasa[2]

Abstract
Region growing is a simple and fast region-based approach to image segmentation. This algorithm exploit the fact that pixels closed together have similar grey level, and starting from one or more seeds iteratively aggregates neighboring pixels to extract meaningful objects. In this work we proposed a 3D implementation of region growing method to effectively segment image data volumes. In particular, we applied the 3D algorithm on Chest Computed Tomographies (CT) from patients with COVID-19 infections(*1*).

Method implementation

The workflow of our algorithm, implemented in MATLAB, can be divided in four main steps:

1. Image acquisition
2. Edge detection
3. Seeds extraction
4. 3D growing implementation

We loaded in MATLAB a 3D image downloaded from the Covid19 Data Portal. We enhance the contrast of grey levels adjusting image intensity values for a better visualization.

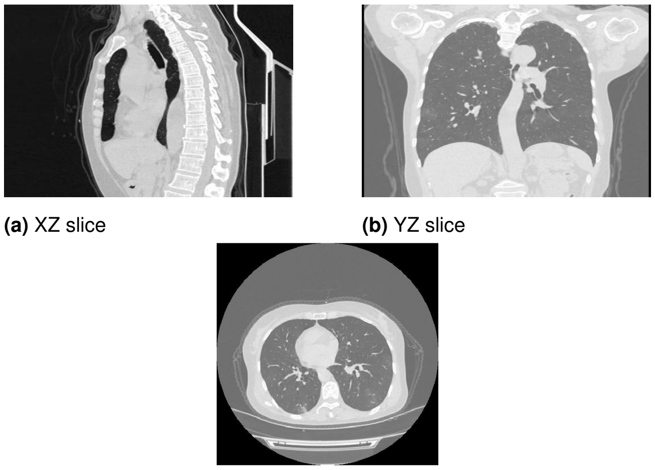

(a) XZ slice **(b)** YZ slice **(c)** XY slice

Figure 1. 3D image downloaded

Due to the huge size of 3D image (512x512x384) and the subsequent time consuming process, we decided firstly to cut the image by selecting the most interesting and representative region.

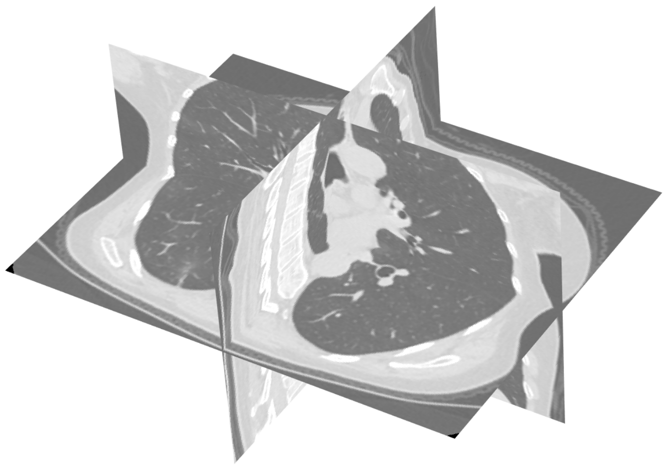
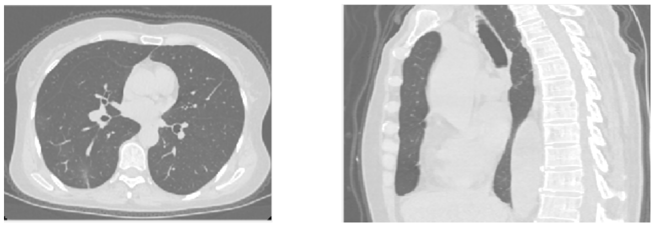


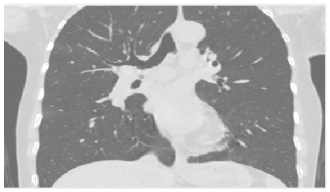
Figure 2. 3D small image (330x220x220)

The second step consists of the detection of image edges. In order to obtain a better result, we first applied the Gaussian and the box filter, to blur the image and remove some contours for a more accurate edges detection. Then we applied the edge3 function (implemented in the Image Processing Toolbox) to detect image edges and we cleared the borders.

[1]University of Trento, MSc in Quantitative and Computational Biology
[2]University of Trento, MSc in Computer Science

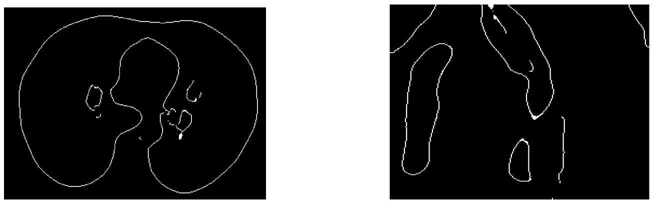


(a) XY slice **(b)** XZ slice

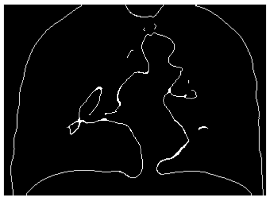


(c) YZ slice

Figure 3. 3D small image



(a) XY slice **(b)** XZ slice



(c) YZ slice

Figure 4. Image edges

For the extraction of the seeds we extended a previous seeding algorithm implementation for 2D images, provided by Oscar Tonelli and Paolino Virciglio. The purpose of this algorithm is to extract seeds placed in the middle of the regions bounded by the edges that we previously found. In this way we were able to detect a large number of seed points needed for the growing algorithm. To avoid oversegmentation we decreased the number of seeds, trying to remove those points belonging to the same edges-bounded region. Furthermore, we removed those seeds that were to close to each other, setting a minimum distance threshold between points of 10 pixels.

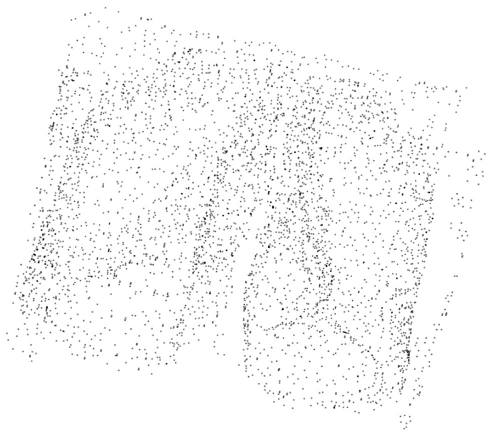
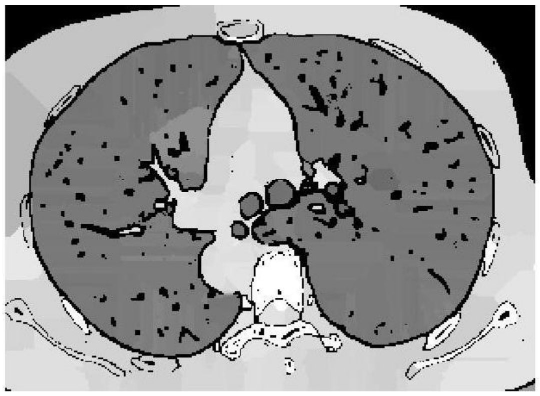
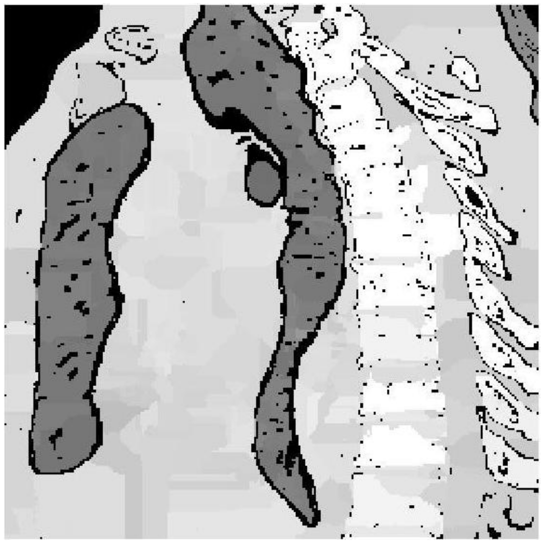


Figure 5. Seed points

The fourth and last step is the growing algorithm. The idea behind it, is to iteratively look at all the seeds extracted in the previous step, and for each of them explore their neighbours and compare the grey level. If the intensity difference between the neighbour and the seed is below an user-predefined threshold, the neighbour is assigned to the same region of the seed and then it is added to the list of seeds to be processed. To implement the algorithm we used a queue data structure, that was at first filled with the starting seeds, and then at each iteration a seed is removed and processed. Each time we found a new pixel to look at we add it to the queue. The growing process is repeated until the queue is empty. We implemented the growing function in such a way that the user can decide if the algorithm should look at the 26 or 6 connected neighbours of a pixel, and also choose the intensity threshold to be used. Furthermore, we added the possibility to create a mask that allows processed pixels that belong only to the region specified by the mask. In addition, a maximum number of iterations can also be specified by the user, in order to prematurely stop the growing process.


(a) XY slice


(b) XZ slice

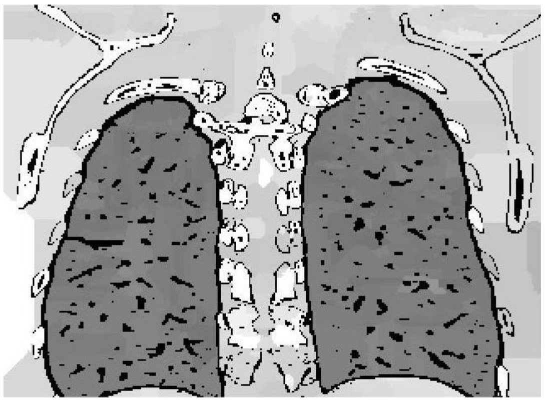
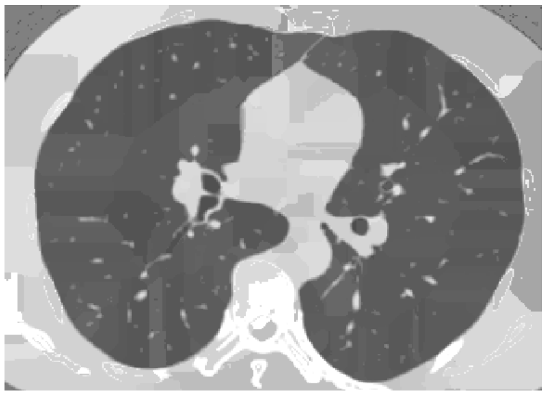
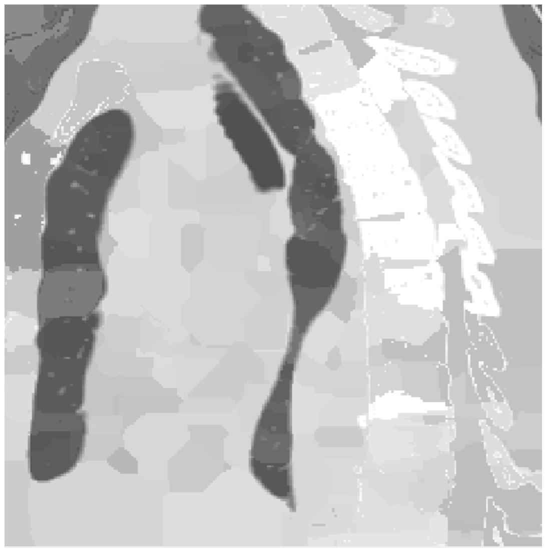

(c) YZ slice

Figure 6. Results of growing algorithm with threshold=3

Since not all pixels of the image are processed, due to the lack of seeds for some regions, we implemented a procedure to process them. We first created a boolean mask, marking the unprocessed pixels, and from these founded regions we extracted the contours to be used as seed points. Then we applied the growing process starting from the new found seeds, specifying also the mask, to be sure that the algorithm grows only in the unprocessed region. This process is repeated until a number of maximum iterations is reached, or until all the pixels are correctly processed.


(a) XY slice


(b) XZ slice

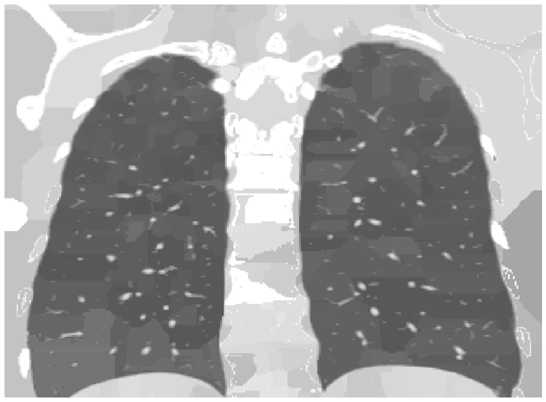

(c) YZ slice

Figure 7. Results after unprocessed pixels filled

Future works

The computational complexity is moderate but the amount of data that need to be checked during the execution of such algorithm on Chest Computed Tomographies is very high. This data increases the execution-time exponentially, a single image requires some hours of computation to be analysed correctly. Luckily, there are many ways to increase the efficiency of such computations. One of the solution that we recognized consists on the parallellizzation of the code, since the pixels of the image can be checked independently and don't required to have and independent and exclusive access. Another possible solution to increase the efficiency is to modify the code and make it run through GPUs, that are more specialized on this type of workload. Furthermore, we have thought about the possibility of recreating the code on cloud services like Google Colab to automate the parallellization and the amount of resources that should be allocate to have a smaller computational time.

Pseudocode

Region Growing 3d

```
"""
 @seeds the matrix of initial seeds
 @intensity_matrix intensity of pixels
 @mask part of the image of interest
 @thr the similarity threshold
 @mode 26 neighbors or 6 neighbors
"""

# define and init output matrix and Queue
m,n,k = intensity_matrix.shape()
out_matrix = new matrix[m,n,k]
q = new Queue()

# find cell's coordinates of all initial seeds
s_x,s_y,s_z = where(seeds == TRUE)

# insert seeds into the Queue and assign to the
# output matrix the same intensity of the seed
for i in range(0, s_x):
  if (mask[s_x[i], s_y[i], s_z[i]] == 1):
   x,y,z = s_x[i], s_y[i], s_z[i]
   q.push([x,y,z])
   out_matrix[x,y,z] = intensity_matrix[x,y,z]
  Endif
Endfor

while (not q.isEmpty()):
  # extract from the Queue and get his neighbors
  p = q.pop();
  neighbors = get_neighbors(p, mode)
  # seed intensity
  s_int = intensity_matrix[p[0], p[1], p[2]]

  for n in neighbors:
   x,y,z = n;

   # check if it satisfies boundary conditions
   if (0<=x<m and 0<=y<n and 0<=z<k # in borders
     and mask[x,y,z] == 1           # in mask
     and out_matrix[x,y,z] == 0):   # first time

     # extract the neighbor intensity
     int_n = intensity_matrix[x,y,z];

     # check if in the threshold
     if(s_int - thr <= int_n <= s_int + thr):
       # add to seeds, set the same value of seed
       out_matrix[x,y,z] = out_matrix[p]
       q.push([x,y,z])
     Endif
   Endif
Endwhile
```

References

1. Covid19 Data Portal (Accessed: 2021-05-08)