Image segmentation using Region growing

DESBIAUX Arthur, SEN Abdurrahman, VADUREL Benjamin
Université Claude Bernard Lyon 1

Abstract

This document is the report of our lab work on the subject of Image analysis, in particular the process of Region Growing in order to segment a given image into multiple regions for the purpose of image analysis.

1. Introduction

This document is the report of our lab work on the subject of image analysis, in particular the process of region growing in order to segment a given image into multiple regions for the purpose of image analysis.

1.1. Language

The code base has been implemented in C++, using the library OpenCV.

1.2. Code repository

The repository used for this project is available at the following address :
https://forge.univ-lyon1.fr/p2006393/m1-analyse-image

1.3. Installation

For the installation process, please refer to the README Markdown file available at the root of the repository.

2. Experimental Steps

Image segmentation can be broken down into the three following steps :

1. Seed selection
2. Region growing
3. Adjacent regions fusion

The first step, Seed selection, determines where to plant seeds on the image.
There exists multiple methods for this process, we will present the one we have chosen in the following section 3.

After the seeds have been planted, they each need to grow and agglomerate neighboring pixels given a chosen similarity criteria, we will elaborate on the process and the criteria we have decided upon in the following section 4.

The last step in the process happens after all the regions have grown as much as they can. Adjacent regions that respect a given similarity criteria have to be fused or merged into a single region.
The method employed and the similarity criteria that we have picked will be developed upon more in the following section 5.

3. Seed selection

The term seed refers to the original pixel from which a region will be allowed to grow.

In order to plant the seeds, we decided to subdivide the image in a number of smaller areas and randomly select a pixel within each.

The subdivision is using the user defined number of seeds and the input image width and height to determine coordinate intervals within which we randomly select a pixel as the seed.

3.1. Representing the regions within the image

Each region is assigned a unique integer as an identifier. We represent our input image as a two-dimensional array of integers, hereby named **regionIdArray**. This array is shared between every region so as to allow easier communication between regions and faster computing.

The array is initialized with the default value -1 for every cell in order to represent pixels of the image that have not yet been claimed by a region.

3.2. Planting the seeds

When seeds have been generated, we return a list of region where each region is only as big as a pixel, their seed, that will be used to process the region growing.

We create this list inside a factory class, which goes through the following steps :

- Calculate intervals within which a seed can be planted
- Shuffle a list of indexes ranging from 0 to the $numberOfDesiredRegions - 1$

- Randomly select a pixel within each interval
- Create a region from this seed and push it into the list

The subdivision of the image width and length is done using this formula :

$$round(numberOfRegionsDesired/2)$$

This allows us to compute a smaller amount of potential intervals compared to simply dividing dimensions by $numberOfRegionsDesired$.

4. Region growing

In order to grow regions, we have two steps : expand the region by searching for pixels to add in the region and compute the border.

4.1. Expand the region

The grow function will iterate on each region until no pixel is added. On each iteration, we loop on all the last pixels we added and search for the 8-neighbor pixels. We can then loop on all the neighbors and verify several criteria:

- Is the pixel already studied ?
- Is this pixel already in another region ?
- Is the color of this pixel is similar to the color of our region ?

To know if a color is similar to another we check if it is contained within an user-defined range. More specifically, we compute the sum difference of RGB values of the pixel with the region's color. If that value is below the user-defined threshold, we believe the similarity criteria to be respected and thus we can consider including the pixel inside the region.

In order to avoid processing a given pixel twice, we use a two-dimensional array of boolean values to know if we have already studied a pixel; by doing that, we gain in performance because we're not going to make redundant calls to previously computed functions

If all of those criteria are checked, we can add the pixel to our region.

4.2. Compute the border

Firstly, we use a vector of pixels (x, y pairs) to represent the border of a region. After this, we loop on all the pixels of the image and verify if we have already studied this pixel.

Assuming we did, we then check that it is in our region, then we check for the adjacent 4-neighbor pixels. If all the 4-neighbors are in the region this means the pixel is not on the edge of the region, and thus not part of the border.

5. Adjacent regions fusion

To merge regions, we proceed in 3 steps : computing neighbors for every regions, searching for a neighbor region with a similar color and if 2 regions match then we fuse the 2 regions.

5.1. Computing neighbors

In order to merge regions, we need to find those that are adjacent. For a given region, we are looping on each border pixels and fetching region identifiers of 4-neighbors pixels. As computing region neighbors is a heavy process, we are storing computed neighbors in the region data structure, and then only updating neighbors for merged regions.

5.2. Merge criteria

To be merged, two regions need to be adjacent and we are checking that region colors are similar enough. Similarly to region growing, we look at each components of the color (red, blue, green) if they are within an given threshold. This threshold used to merge regions is stricter than the one used to grow regions.

5.3. Merging two regions

Once we found a region to merge into another, we first update the second region's pixels with the first region identifier. Then we update neighbors for both regions' neighbors as the second region no longer exists. The merged region will get a color corresponding to the average of the two regions' colors and an updated border. Then, the algorithm will search for other regions to merge.

6. Region growing on videos

The algorithm used to process a video is a simple loop on each frame of the video on which we compute a segmentation just like a simple image. Once every frame have been processed, we put them all together in a new video.

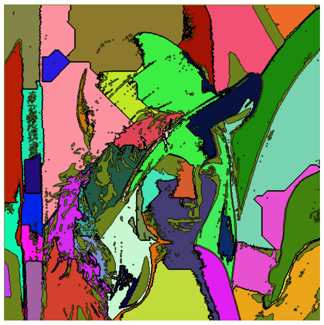
7. Result

7.1. Sequencing

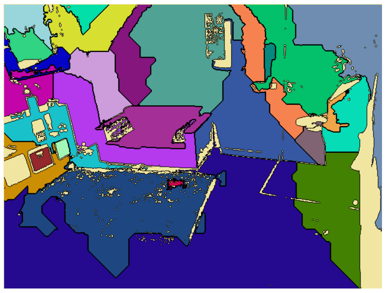
Here are some results :

7.1.1 Image

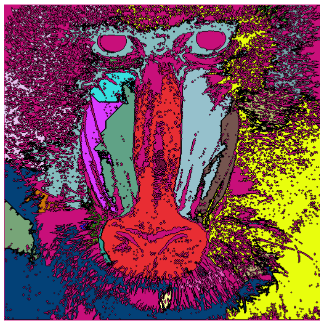
(a) Original image Lena


(b) New image Lena

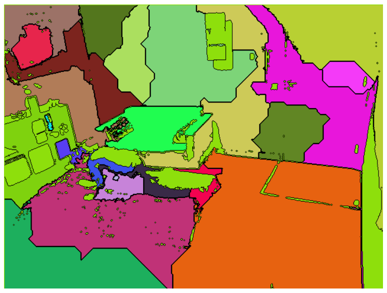
(a) Original image room


(b) New image room

(a) Original image baboon


(b) New image baboon

(a) Original image room2


(b) New image room2

7.1.2 Video

You can download the resulting video available until 13/02/2024

7.2. Performance

This section is about the time of execution we have on every image For all our test we use 100 germs and 10 threads

7.2.1 Image

- lena : 2,343s
- baboon : 2,787s
- room : 2,967s
- room2 : 3,201s

If we try to increase the number of germs, we are going to increase the time of execution. For example on room2 with 400 regions with have 6,904s

7.2.2 Video

For the test video, it takes us around 14m4,773s to compute segment each frame of the video and create a result video. The duration of the original video is 30s.

For a 5s duration video we have 54,458s of execution.