Bishop’s University

CS463 winter 2023

Computer Vision-Image processing

Instructor: Dr. Elmehdi Aitnouri

Assignment management

Number: 3

Name: Analysis of binary images

Team member

Last name	First name	ID	Email
Sparnaay	Jerome	002320723	jsparnaay21@bishops.ca

For this assignment, one needed to create a program to cluster using k means. This is used to segment the image based on the proprety that some pixel might share similar values such as a similar position in the image or their color/gray level.

To make this, new code needed to be made to allow the program to use color images, which were not needed in the previous assignments. To do this, a new struct type was made with the 3 Img struct pointer. Then when putting the image in the buffer, each color is treated as its own image which can be modified separately and all have the right position index. With little modification, one could use previous functions with a per color manner.

Next, another new struct type was created to ease calculations of the vectors. This being the ListPoint, which contains a **double with the size of the list and the size of each array contained -2 . This -2 is to account for the position ot the pixel and the rest size of the array (pointSize) is therefore the number of values for the color of the pixel. Although applied in the grammar, The PointList.list is exclusively used with the function Point() to set each point of the list by passing it the values of pixels color/grayscale and its position. This insures that the position is always at index 0 and 1, followed by the value of the pixel. This is used include or exclude the position in the means calculation. The program asks once for a int to include or not the position and this is passed on in every functions that requires it. This int ends up in the function that calculates the distance between points and if it is not 0, will set the loop to calculate distance with an initial position of 2. This effectively makes the calculation ignore position. The minimum distance that is used to calculate in which cluster a point is is simply a loop that finds min and calculates the distance of a pixel point with list of points and returns the closest points index.

Another consideration that was made was in the color of the output pixels. They are simply the averaged value of all the pixels in the cluster.

With for simplicity, no multi threading was implemented and including them could’ve broken the compatibility with windows machines (this being developed in linux). This lead to the execution to be noticably slow when many calculations when certain variables are set high. In order of change from big to small, the number of images out (repeats all steps this number of time), the number of generations (need to go through each pixel before iterating, 50-350 is usually enough to yeild a minimum), the number of clusters and if it is color/withPos/grayscale. Since the initial random clusters can result in better of worst images, the program can output multiple images with the same options and different random initial cluster each time. Each image will have the name “out%d.ppm“, i. Watch out though, since each image can take a few seconds to minutes to run. Finally, some data needed to be allocated using malloc and is only freed once the program exits normally, although the os should deallocate

To increase the speed of execution, one can uncomment line 5 of CmakeList.txt. In practice, adding O3 optimization compiler optimization resulted in a noticeable decrease in execution time.

Results

For results, they can be separated in 2 categories, one with the position of pixels being considered and one without. That is the ones with position. The ones with position all have the issue that the solid background colors needed to have multiple centroid to not interfere with the subject of the image, which also had the consequence of having blocky backgrounds leading to worst segmentation. On the other hand, the ones without positional values did significantly better on segmenting. Also, making the the kmean only rely on color meant that more details seemed to have been kept and using a rather largeer number of kmean, some test were performed to see if this could be a good method of image compression and looking at some results, it seemed to performed well. This is because instead of storing 3 * 1 byte per pixel, a single byte could've bee used to store the color of each pixel with a map of what each value could be. Another way to theoretically compress it even more would be to compress the image to large number of centroids with positional value, eliminating the usage of the original image completely. Doing quick math, using around 8000 centroids (as in the photo), we are using 8000*(8 for double + 3 byte for color) 88000. this is just over a third of the 256 no position one, although many artifacts are present. To decompress, simply find what centroid is closest to each pixel and assign the value.

Lena 8000 centroid with pos

lena256

centroid no position

lena original

plane with pos

plane without pos