Assignment 4: Template search in Image using CUDA

Problem Statement:

In this assignment, you are required to use CUDA for parallel computation. You are not allowed to use any frameworks. Instead, work on CUDA (version CUDA 11) in C++.

You will be given a Data RGB image (call it L) and a small query RGB image (call it Q). Your task is to locate the query image Q approximately in the data image L. Note that the query image Q need not be upright with respect to the data image L. There may be a rotated copy of query image Q in data image L. Thus, a match is specified by the X, Y pixel coordinates of the lower-left corner of the query image Q in the data image L and its counter-clockwise rotation in degree of the base of query image Q. To simplify the problem, we will only rotate the query image from -45° to +45° in steps of 45°. Image coordinates are (0,0) on the lower left. The required output is a series of <X, Y, degree> triplets.

The image will be passed as a text file. The first two space-separated integers would represent the number of rows (m) and the number of columns (n). Then m*n*3 space-separated integers will be given that would represent the coordinate at Image [i,j,k].

<M:int> <N:int>
R G B R G B (M*N times) -> Reading row by row

So for array A of m*n*3 integers, X[i,j,k] = A[i*n*3 + j*3 + k]. That means it would be reading images row by row and writing each pixel as a triplet of R G B channel values. Pixel value that is X[i,j,k] lies in range [0,255] with integers value only.

A perfect match is found if the pixels of the query image Q match the pixel values of the data image L exactly. Note that pixel coordinate <X, Y> of the query image may not have integer coordinates after rotation by d degrees. You compute the colour of non-integer pixel locations of an image using bilinear interpolation. Read bilinear interpolation.

We use the interpolated data pixel to match against each query pixel in the case of a rotated query image. We are looking for similarity, and not necessarily a perfect match. This means that RMSD (root mean square of the differences) between two images is less than some given threshold.

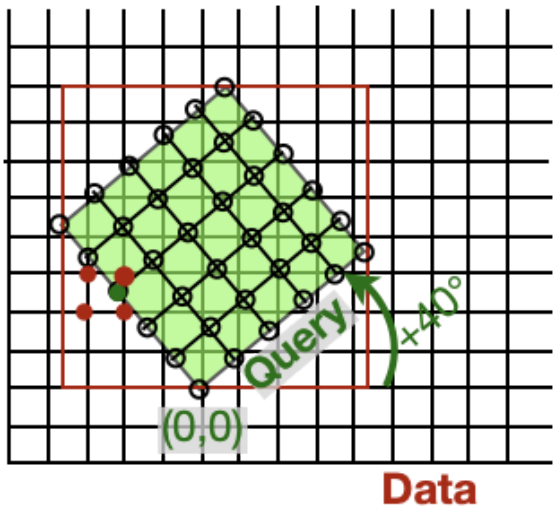
One can read in detail about RMSD. RMSD is the root of the mean of the sum of squares of the difference between corresponding pixel values for each channel (R, G, B).
If the image size is m*n*3 where 3 represents RGB channel then RMSD is given by:

$$RMSD = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{3}(X[i,j,k] - Y[i,j,k])^2)/(m * n * 3))}$$

The brute force method may be too time-consuming. One can always filter out the image on basis of some basic condition. There are several filtering techniques. We will use a very simple one. Convert each image to grayscale by taking the grey value V to be (R+G+B)/3. Also for filtering, we can compare with the upright bounding rectangle of the query image (as one can see a red square in the image below). In the case of the rotated query image, we use its axis-aligned bounding box in the data image to filter. Only if the average of all grey values of the bounding box image is within TH2 of the query image (rotated and interpolated), shall we check if the RMSD is within TH1.

An image summary can be computed to represent areas of an image. If two images do not have a similar summary, they may be considered different enough and the detailed RMSD computation may not be necessary. In our case image summary would be the average of all pixel values means it would be an integer.

Filtering method:

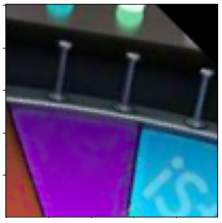


In the above figure, each grid location stands for a pixel. The green rectangle represents the query image rotated by 40°. Its bottom-left corner (coordinate 0,0) is aligned with some pixel of the data image (represented by the upright grid). The filled green circle of the query image is compared with the interpolated value from the four dark-red data pixels. The average data grey-scale values in the bounding box marked in dark red are used for filtering. Any data pixel on the boundary of the bounding box is included in the average.

Note: In the example above angle is given at 40 degrees. But for our testing, we will use only three angles -45, 0 and 45 degrees.

Example:

Data Image (L):



Query Image (Q):

For the above image, for n=1 output triple would be (290,120,45) given the data image size is (410*552) and the query image size is (100,100). One may notice that it is possible to not get integer coordinates and hence bilinear interpolation is used for getting approximate values.

Evaluation Scheme:
We will check for each given data image, query image, threshold 1, threshold 2 and ‘n’, the triplet outputs. We will see if RMSD is within the threshold and is in top n.
In this assignment, if you give the topmost match (n=1) you will get full marks. If you give top ‘n’ then there is a 15% bonus.

Deliverables

- A zip archive with the filename <entry_num1>_<entry_num2>.zip. On unzipping it should produce a directory with the name as your <entry_num1>_<entry_num2> (all in caps).
- The directory should contain the make file, a bash file and all other files to run your code. Do not refrain from this format.
- First, we will run the make file. It should give executable. You are required to write a run.sh bash file which will take executable and arguments and write top ‘n’ triplets in output.txt.
- We will run bash file as follows: ./run.sh <path_of_data_image> <path_of_query_image> <threshold_1> <threshold_2> <n as in top n>
- We will be using CUDA 11 in our HPC testing system. Please ensure it runs for our testing system.