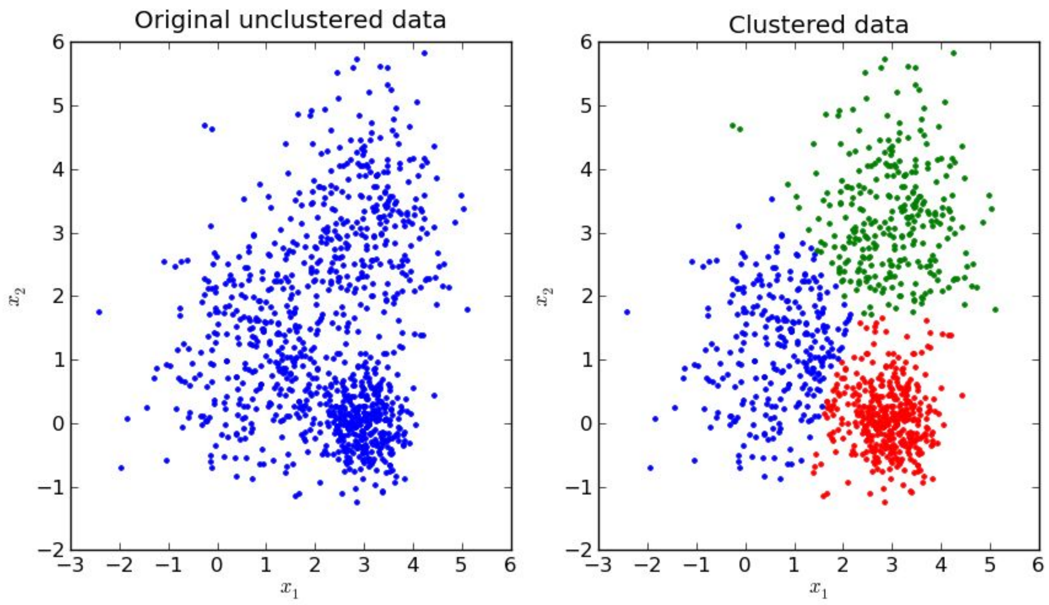
K-Means Clustering Algorithm
Graphic Processors in Computational Applications

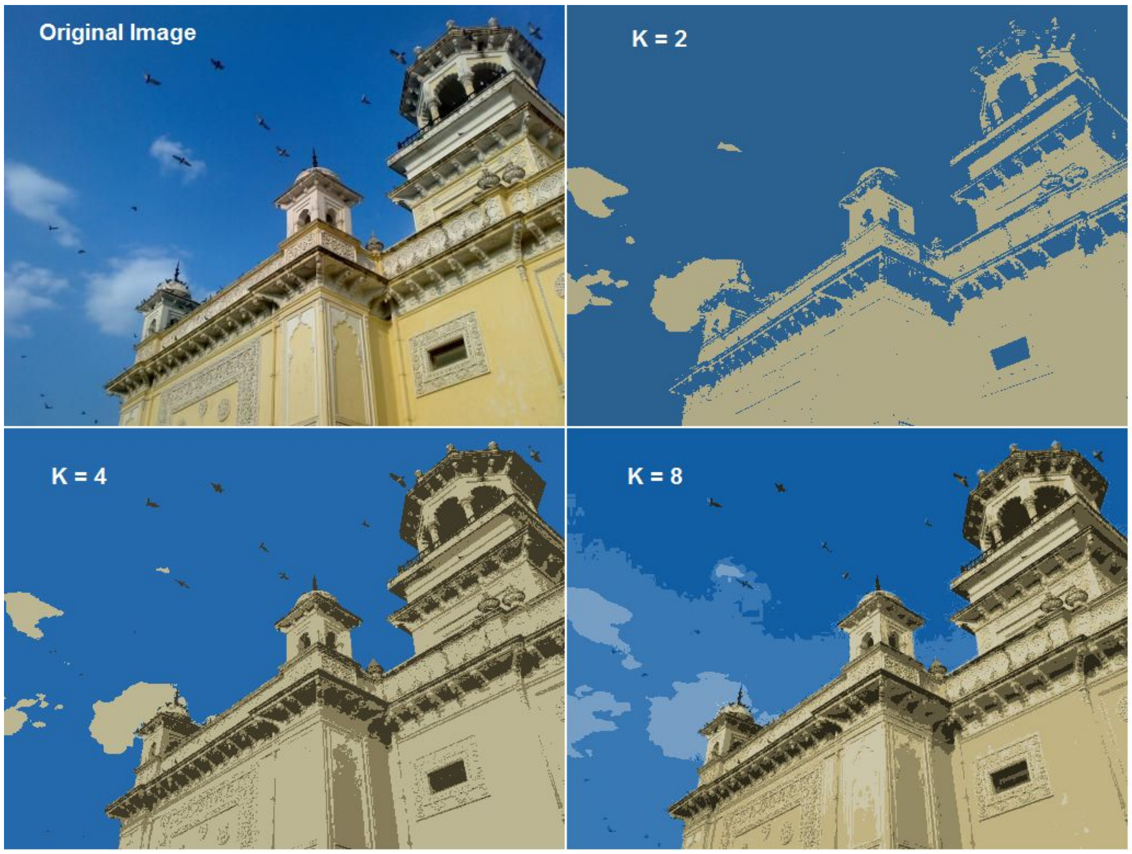
Filip Matracki



Introduction

The k-means clustering algorithm is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. For example, K-means clustering can be used in Computer Graphics in order to reduce the amount of colors in an image while preserving the "quality" of the image. We can take an image of n pixels and partition the color space of the image into k clusters. Once that is done, we can assign each pixel the value of the cluster it is a member of.

Example:



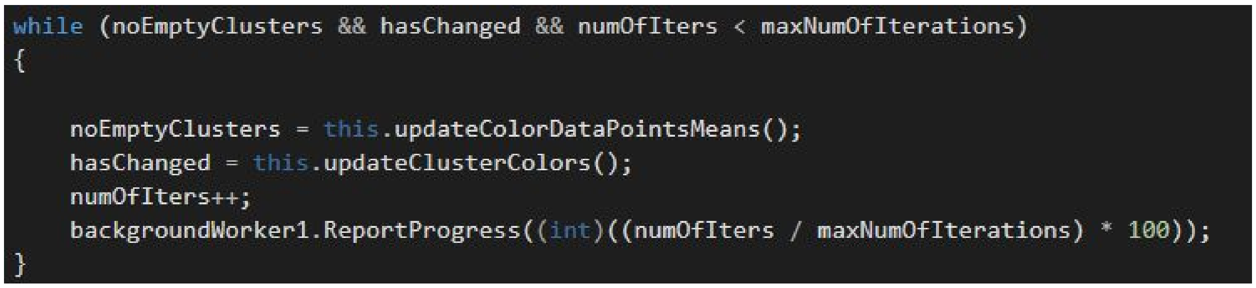
GPU Implementation

The general algorithm is as follows:

1. We initialize k-means data, and allocate memory on host and on device
    a. Our cluster centers are initialized to the first *numOfClusters* elements in our data set
2. We call *findNearestCluster()* kernel function, which finds the distance between each cluster and a data object, the data object joins the cluster with the smallest distance. The GPU parallelism exists in the fact that individual thread finds this distance at the same exact instant. It is not necessary to sequentially go through all the objects like in the CPU implementation.
    a. If the new cluster is different it sets the *membershipChanged[threadIdx.x]* value to 1
3. We call the *computeDelta()* kernel function which performs array reduction (summing all elements in the array) in the same way as in Mark Harris' array reduction presentation found here:
    a. http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf
    b. We sum the amount of data objects that have changed their cluster
4. On the host side we find the new averages of the clusters (the new clusters). We do this on the host side because this step is difficult to parallelize.
5. We go back the step 2 and continue until the delta (found in *computeDelta()* is less than *threshold*. Both *delta* and *threshold* are values between 0 and 1. If *delta* is 0.5 it means 50% of the data objects have found joined a new cluster in this iteration
6. We output the file

CPU Implementation

The CPU implementation is shown below, this is the code found in my CPU implementation that I used for my Computer Graphics course in image processing.



```
while (noEmptyClusters && hasChanged && numOfIters < maxNumOfIterations)
{

    noEmptyClusters = this.updateColorDataPointsMeans();
    hasChanged = this.updateClusterColors();
    numOfIters++;
    backgroundWorker1.ReportProgress((int)((numOfIters / maxNumOfIterations) * 100));
}
```

Where *updateColorDataPointMeans()* does the same thing as point 4 in the GPU implementation, and *updateClusterColors()* does the same thing as points 2 and 3 in the GPU implementation but sequentially instead of in parallel.

How to run the GPU and CPU programs

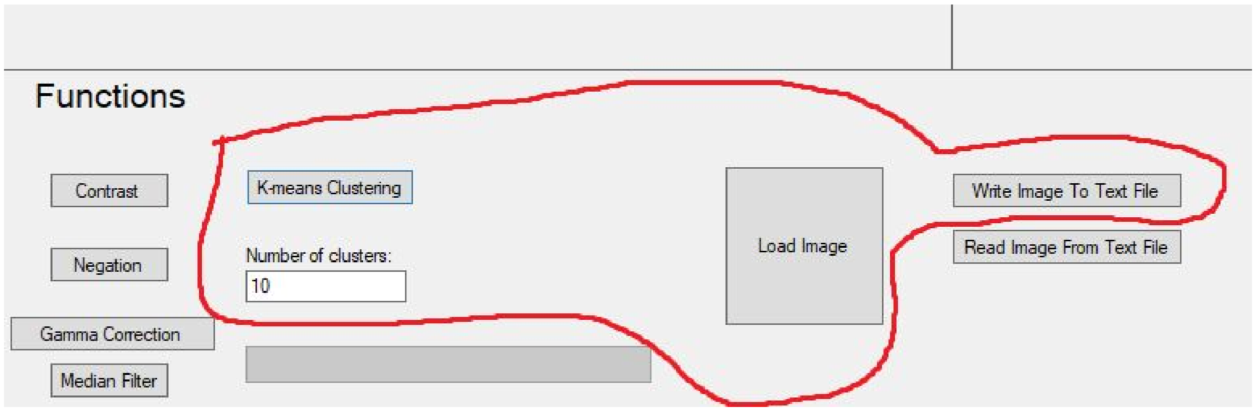
GPU

1. The first program argument is the name of the file that has this format:
    a. ASCII text format
    b. Should be located in the same directory as the example *images.txt* file that was attached
    c. Each line contains the line number (starting at 1)
    d. After the line number we have the coordinates of the data objects (for example 122 100 255 for an RGB color)
    e. Each line must have the same amount of coordinates
    f. See the example image.txt file attached

2. The second program argument is the number of clusters k.

3. After the program is finished executing we print the total execution time and output 2 files:
    a. Coordinates of cluster centers
        i. The file name is the input file name appended with ".cluster_centres".
        ii. It is in ASCII text format.
        iii. Each line contains an integer indicating the cluster id and the coordinates of the cluster center.
    b. Membership of all data points to the clusters
        i. The file name is the input file name appended with ".membership".
        ii. It is in ASCII text format.
        iii. Each line contains two integers: data point index (from 0 to the number of points) and the cluster id indicating the membership of the point.

CPU

1. Done in Windows Forms for my Computer Graphics course
2. Click on the LoadImage button to load an image
3. Press the K-means clustering button after selecting the number of clusters
4. A dialog will appear that will give results of the execution
5. The only buttons that are necessary to press are shown below:
6. 

7. **Writing image to a file will produce a result that is compatible with the GPU k-means image format!**
    a. (Reading image from text file is not supported at the moment), only from a standard image file)

Performance Analysis

K = num of clusters

For file image.txt (attached in the zip file):

- **CPU**:
    - K = 5: 1.02s
    - K = 6: 5.03s
    - K = 7: 11.1s
    - K = 8: 7.9s
    - K = 9: 14.3s
    - K = 10: 15.2s

- **GPU**
    - K = 5: 0.116s
    - K = 6: 0.124s
    - K = 7: 0.105s
    - K = 8: 0.08s
    - K = 9: 0.1s
    - K = 10: 0.105s
- Notice how the CUDA implementation is actually faster for larger k!