Christian E. Merchant & Mateusz Czarnocki

Due: 12/5/14

CSCE 350 : Final Project Write-up

Dr. Tong

**Introduction**

The goal of this project was to select the images from a template “training” set that could then be used in a k- nearest neighbor (k-NN) implementation for a given query image. In this project, each image in both the query image and training template have features sets of 5,632 elements. The training template contains 138 individual images stored as ASCII RGB values. The similarity was calculated using a cosine function in which the dot product of the query image and the template image was divided by the multiplied magnitude of the query image and the template image:

**Problem Solution**

As per the project description, we used the proposed formula for computing the similarity. The stepwise process for achieving a correct solution was as follows. We chose a monolithic design as we felt it would eliminate extraneous function calls, thus saving time:

1. **Timer:** Start the execution timer
2. **Data Input:** Create arrays for all query and template names
3. **Data Input/Compute Similarity:** Open input stream and read in three queries sequentially for every template image. Compute the similarity for each query file with the appropriate template and store in a similarity vector of doubles. As stated, we used the formula in the provided project description. Initially, we had separate functions for computing the dot product and the magnitude. We were able to shave off a few tenths of a second by combining these functions into the similarity function. While filling the similarity vector, we created an index vector with the index values from the similarity table
4. **Sort:** Sort the index by referencing the similarity values using a proprietary quicksort function. We chose to sample a pivot from the middle of the image feature set as many query and template files contained zeros (see below for improvement suggestions).
5. **Print:** Until the similarity vector of every query had been calculated, print to the logfile the ten nearest neighbors for the given query (i.e. the last ten values in the index vector) as well as the corresponding similarity values.
6. **Timer:** Print execution time to the logfile (see separate heading for performance)

**Results (Execution Time):**

Linux Lab Computer: Hickory

Total Execution Time: 9.72299 seconds

Time Efficiency of Solution: Ω(n4)

**Other Efficiency Improvement Suggestions and Literature Review**

Several areas of improvement are suggested with this project. It should be noted that image matching is a vast area of study in computer science. Many different methods have been proposed as improvements with their own drawbacks and advantages. An example drawback to k-NN is that it requires large template sets to satisfactorily resolve a query image. As the instructor mentioned, our project is but a small subset of the actual research data. This indicates one widely used solution to this k-NN drawback – training set size reduction. We noted that the time required for our particular solution was mostly taken up with data input and storage in data structures. Disk access is exceptionally expensive from the perspective of time complexity. We postulate that if one could perform the necessary computations directly from the input stream or could develop a multithreaded application, we could see great performance gains from our current execution time. The addition of preprocessing steps like analyzing the training images to further reduce their number could also be a productive activity:

* It was noted that the data files themselves had many zeros. As quicksort was used, a pivot of zero would yield the worst case (O(n^2)) for sorting is such instances. A helper function could be employed that would randomly sample the feature sets for a non-zero pivot. This would require additional time, however, and it may have been more efficient to switch methods to mergesort or perhaps insertion sort as the similarity vector is built.
* A method for presorting the feature sets themselves could also be useful. While we only compute the similarity for those values that are non-zero, if the feature sets were presorted, magnitude computation would be simplified as all the zero values could be thrown out – only a portion of the feature set would need to be used.
* Another proposed efficiency suggestion would be to break out of the similarity computation when it is determined that it was smaller than the ten nearest neighbors. The current ten values could be stored as a min heap and the new similarity value would be checked against the minimum value during the computation
* Cutting down the feature set using a different method of measurement and then using a fast approximation algorithm with a low miss percentage would also enhance the speed of this process. Building up a similarity value is an expensive process using a brute force implementation.
* One could also conceive of a process by which template sets represent nodes in a binary tree implementation. Finding the k – nearest neighbor would could be computed at each right subtree
* Scholarly methods for fast image similarity computation abound. One potential solution is to use a fast approximation algorithm that yields the same results as our implementation but with comparatively lower overhead (Durand, et al. 2012). This method can be viewed as an extension of our nearest neighbor implementation. Other solutions would require a different method of measuring image data instead of the traditional RGB features. Image decomposition and selection of feature vectors allow Haar wavelets to be used to identify images similarity. Ferns and keypoint recognition can be employed to decrease the amount of necessary preprossessing and scales efficiently to larger feature sets (Özuysal, et al. November 2008). The enhanced throughput and parallel computing afforded by the ALUs on a GPU have also been used with success in k-NN search as well (Garcia, Debreuve and Barlaud 2008). It should be noted that the authors could not use quicksort in the GPU implementation because CUDA cores do not permit recursive functions.

**Group Participants and General Assignments:**

Mateusz Czarnocki – Data I/O, Code Formatting, Driver (shared), efficiency improvements (shared), Code Comments/Documentation (shared), scripting

Christian Merchant – Data Manipulation: quicksort with indexing, similarity computation, dot product, magnitude, Driver (shared), efficiency improvements (shared), Code Comments/Documentation (shared), Write-up, ReadMe

**Pseudo-Code:**

//MAIN METHOD

execution timer <- start

Array queryNames <- query file names

Array templateNames <- template file names

output <- Open Logfile

for (0 up to total number of total number of templates)

for (0 up to images per template)

while (there are values to add)

vector image < - values

double vector template <- template images

for (0 up to three queries)

while (query file has additional values)

vector query <- values

for (up to total images in each template)

vector similarity <- compute similarity (query and template image)

vector index <- index number of each image in template

clear template image

clear query

quickSort (index using similarity values)

output <- last 10 values in index vector and corresponding similarity value

clear index and similarity

update start pointer in queries vector

clear template

execution timer <- end

output <- execution time

(End Driver)

//SIMILARITY FUNCTION

Method to compute the similarity using provided formula in class by computing dot product and magnitude. Note that we make this more efficient by not considering values in the template or query feature set that are zero – they won’t contribute to the similarity calculation

//QUICKSORT

Method to sort the index data structure by indexing into the similarity data structure. Pivot is taken from the middle of the feature set to limit the number of pivots that have a value of zero

# **References**

Durand, FrØdo, Michael Gharbi, Tomasz Malisiewicz, and Sylvain Paris. 2012. "A Gaussian Approximation of Feature Space for Fast Image Similarity." Technical Report, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA. http://people.csail.mit.edu/tomasz/papers/gharbi\_techreport\_2012.pdf.

Garcia, Vincent, Eric Debreuve, and Michel Barlaud. 2008. "Fast k Nearest Neighbor Search using GPU." Technical Report, Universite de Nice-Sophia Antipolis, Sophia Antipolis, France. http://nichol.as/papers/Garcia/Fast%20k%20nearest%20neighbor%20search%20using.pdf.

Özuysal, Mustafa, Michael Calonder, Vincent Lepetit, and Pascal Fua. November 2008. "Fast Keypoint Recognition using Random Ferns." Technical Report, Computer Vision Laboratory , Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland. http://cvlab.epfl.ch/files/content/sites/cvlab2/files/publications/publications/2010/OzuysalCLF10.pdf.