

ECE 734

PROJECT PROPOSAL

Implementing Memory and Run-Time Efficient  
Texture Classification using  
NVIDIA GPU, as a co-processor.

Shreyas Parnerkar

## **Motivation:**

Classification refers to as assigning a physical object or incident into one of a set of predefined categories. In texture classification the goal is to assign an unknown sample image to one of a set of known texture classes. Texture analysis is important in many applications of computer image analysis for classification or segmentation of images based on local spatial variations of intensity or color. Important applications include industrial and biomedical surface inspection, for example for defects and disease, ground classification and segmentation of satellite or aerial imagery, segmentation of textured regions in document analysis, and content-based access to image databases.

However, a major problem is that textures in the real world are often not uniform, due to changes in orientation, scale or other visual appearance. In addition, the degree of computational complexity of many of the proposed texture measures is very high.

The most successful methods for texture classification are through textons which are cluster centers in a feature space derived from the input. The feature space is built on the output of a filter bank. The process is computationally expensive since the images are convolved with large filter banks and in most cases requires clustering in high dimensional space. [1]

LM filter bank results in a 48 dimensional feature space, while S filter bank results in a 13 dimensional feature space [1].

[1] proposes a new method for texture classification without the use of textons. Tuzel et al. use image intensities and norms of first and second order derivatives of intensities in both x and y direction for texture classification which results in a 5 dimensional feature space.

## **Approach:**

The texture classification algorithm is usually divided into two phases namely training and testing.

In the training phase, random square regions of random sizes are sampled from the training set of images and covariance matrices are calculated. Thus each texture image is represented with these covariance matrices. We form a set of such covariance matrices for all the texture images. This is again done for all classes of textures.

In the testing phase again covariance matrices are computed for randomly selected regions from the test image and are compared with the covariance matrices using a distance metric.

[3] gives a distance metric for Co-variance matrices. [1] and [2] propose that computing co-variance matrices using integral images is an efficient way of computing co-variance matrices compared to conventional histogram matching algorithm.

## **Observations:**

[1] , [2] & [3] together combine to form a complete algorithm for texture classification. But the entire algorithm is not real time and is also a memory intensive implementation.

The features that are extracted from each image form a 3 dimensional feature space.

$$F(x, y) = \left[ I(x, y) \left| \frac{\partial I(x, y)}{\partial x} \right| \left| \frac{\partial I(x, y)}{\partial y} \right| \left| \frac{\partial^2 I(x, y)}{\partial x^2} \right| \left| \frac{\partial^2 I(x, y)}{\partial y^2} \right| \right]^T .$$

**Fig 1**

In order to obtain the last 4 values, first order and second order derivatives need to be computed. This corresponds to the gradient and the Laplacian filtering.

Further this feature space is used to compute the co-variance matrices using integral images resulting in 3 dimensional co-variance matrices.

Though the co-variance matrices using integral images is an efficient way of computing [2], it still takes a large number of cycles in the range of  $10^9$  for a  $256 \times 256$  pixel image and a search region of  $32 \times 32$ . The algorithm proposed in [2] is computationally intensive because of the highly nested loops. The calculation of co-variance matrices is done in both training and testing phase. Hence this part needs optimization both in terms of memory usage and the computation time in order to make it a real time algorithm.

## **Algorithm: Calculation of P & Q Integral Images**

**P --- Integral Image (3D):          Dimensions = rows \* columns \* number of features (F)..... Fig 1**

```
for x in 1 to rows
  for y in 1 to columns
    for index in 1 to featureLength
      for i in 1 to x
        for j in 1 to y
          P(x, y, index) = P(x, y, index) + F(i, j, index);
        end
      end
    end
  end
end
```

**The algorithm for P and Q integral image calculations is an iterative algorithm and also has true dependencies because of cumulative additions (P integral image) and multiply and accumulate operations (Q integral images)**



## **My Approach:**

1. Optimizing integral image (histogram) calculation using techniques learnt in the class. The main emphasis will be on usage of pipelining and parallel processing. For this, dependence graph for the nested loops will be analyzed in order to exploit possible parallelism. Along with this other techniques such as single assignment transformation, loop transformation, look-ahead transformation will be used to further enhance the parallel implementation.
2. Then after the algorithm reformulation, it will be implemented in Matlab in order to do the benchmarking for the runtime of the algorithm and the memory usage of the algorithm.
3. Further the computationally intensive part of the algorithm will be implemented on NVIDIA GPU using CUDA [4] in order to explore its SIMT architecture. This will focus on the filtering of the images for the feature space generation, integral image calculations and the covariance matrix calculations.
4. I will try to provide a memory efficient and time efficient implementation of the implemented algorithm. The size of each pixel is 8 bits as I will be working on gray scale images. The images for "Textures" are obtained from [5] & [6]. I will use the optimum data types at each step of filtering, integral image calculation and the covariance matrix calculations. CUDA provides with a range of data types for efficient implementation.
5. Next, the inherent parallelism available with the GPU programming will be exploited and a time efficient version will be implemented.

## **Expected Results:**

I expect to get a high throughput using GPU implementation compared to complete CPU implementation. Also, I expect to reduce the memory requirement for each of the iteration and reuse the memory in order to increase more parallelism.

I will compare the results with the Matlab implementation on CPU.  
The focus of the project will be on making a real time texture classification.

## **References:**

- [1] Tuzel, Oncel, Fatih Porikli, and Peter Meer, "Region Covariance: A Fast Descriptor for Detection and Classification." (2006): 589-600.
- [2] Porikli, Fatih, and Mitsubishi Electric Research Laboratories, "Integral Histogram: A Fast Way to Extract Histograms in Cartesian Spaces." *Pattern Recognition* (2005).
- [3] Forstner, W., Moonen, B.: A metric for covariance matrices. Technical report, Dept. of Geodesy and Geoinformatics, Stuttgart University (1999)
- [4] NVIDIA CUDA™ Programming Guide.
- [5] <http://www.ux.uis.no/~tranden/brodatz.html>
- [6] <http://sipi.usc.edu/database/database.cgi?volume=textures&image=12>