Face Detection on the GPU

Phil Monroe and Kramer Straube

Mar. 15, 2012

Abstract

In this paper, we implemented the beginnings of the Viola-Jones object detection method as it applies to faces. We also added a single feature classifier to detect the presence of glasses. To implement this on the GPU, we first find the integral image which is done serially because it is faster. Then, we cascade our identifiers and apply them using the integral image to minimize memory accesses. Finally, we present the list of our hypotheses for the location of the face back to the CPU. INSERT SOME JUNK ABOUT RESULTS HERE

1 Introduction

In this paper, we will address the application of face detection on the GPU. We used the Viola-Jones Object Detection algorithm to detect faces. We also added a glasses identifier to make an attempt at detecting whether the faces detected had glasses or not. We tested our algorithm on a subset of the Caltech Web Faces database and a subset of the Caltech 101 image database to analyze how accurate our implementation was. Finally, we discuss the pros and cons of the implementation as well as some notable aspects of our specific implementation.

2 Viola-Jones Object Detection

Viola-Jones object detection focuses on using simple identifiers to determine whether the object exists in the image or not. The algorithm also performs object localization if sufficiently many identifiers are used. We used this algorithm to determine the presence (or lack thereof) of a face in an input image. The algorithm splits the image into numerous *sub-windows* or small regions of the image upon which detection is applied.

2.1 Integral Image

To speed up computation of the identifiers, the Viola-Jones algorithm pre-computes the integral image. The integral image defines each pixel as the sum of all pixels that are both above and to the left of it. We explored both the standard serial implementation of computing the integral image from OpenCV as well as a parallel implementation.

The parallel implementation first performs a scan operation on the columns to create a sum for each pixel location of all the pixels above it. Then, the algorithm does a scan operation on the rows to add up all of the column sums for each pixel to the left of it. This calculates the sum of all of the pixels above and to the left of the current pixel.

2.2 Cascade of Classifiers

The classifiers are simple rectangular identifiers to determine different features of the face. To compute whether the classifier accepts that the given sub-window has a face, the algorithm subtracts the "black" area from the "white" area of the classifier. Figure 1 shows the classifiers we used for face detection. In the Viola-Jones paper, they use two hundred of these kinds of classifiers but we use a small subset of four classifiers because we are interested in how this algorithm applies to the GPU instead of the actual results. Four was sufficient to see the difficulties with implementing this algorithm on the GPU.

The other portion of Viola-Jones face detection was the use of a classifier *cascade*. This means that the output of one classifier (which contains the list of sub-windows that may have a face) is the input for the next classifier. So if any classifier rejects a sub-window, the algorithm rejects the sub-window. If all of the classifiers accept a sub-window, then the algorithm says that there is a face at that subwindow. The alternative to the cascade approach is to apply all of the classifiers to every sub-window and define some number of acceptances that each sub-window needs to determine that there is a face there. We implemented both of these approaches to analyze which would be more efficient on the GPU.

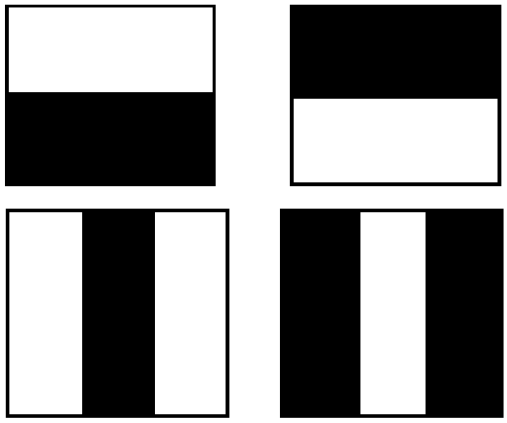

Figure 1: The set of identifiers used in our implementation of Viola-Jones

2.3 Glasses Classifier

text and figure of our glasses classifier

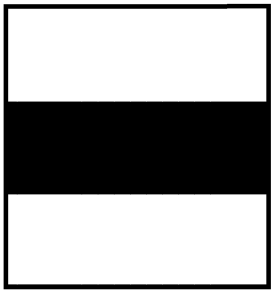

Figure 2: The glasses classifier that we implemented

3 Results

Stuff goes here Actual results: face images: 0/50 glasses 49/50 faces non-face images: 8/50 glasses 38/50 faces glasses images: 20/25 glasses 24/25 faces

Exec time: cascade: 0.020334 brute force: 0.033715 glasses: 0.004104

4 Discussion

64 Threads per block, 3 kernel calls
Max occupancy is at 128 threads per block

5 Conclusion

Conclusion goes here

References

- **EEC 277 Lectures 1+2 plus slides.**
- **David Luebke and Greg Humphrey - How GPUs Work**
  `http://bit.ly/hHt4VH`