Proposal

Summary

I will implement the Viola-Jones facial detection algorithm on an NVIDIA GeForce GTX 1080 found in the Gates machines. I will compare this to a C++ implementation and OpenCV's provided implementation that both run on a standard CPU.

Background

GPU's are very valuable in image processing problems because of the inherent parallelism found in many common image processing problems. Viola Jones Object Detection works by finding Haar-like features corresponding a target image. Haar-like Features are composed of rectangular regions aligned in patterns to find edges, lines, and other primitive features. When these features are convolved over an image, they generate a large response over their represented feature.

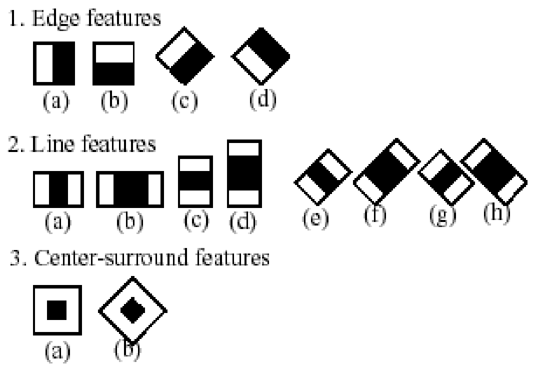

Figure 1: Examples of Haar-like Features

The algorithm works in 2 parts. First is a training step to produce a cascade classifier of features. The cascade is trained with two sets of data: a positive set containing the images with the target object for detection and a negative set containing images without the target object for detection. The cascade will determine which features are common in the positive set. For example, a positive set of human faces should have a distinctive pattern of edge and line responses corresponding to the eyes, nose, and forehead.

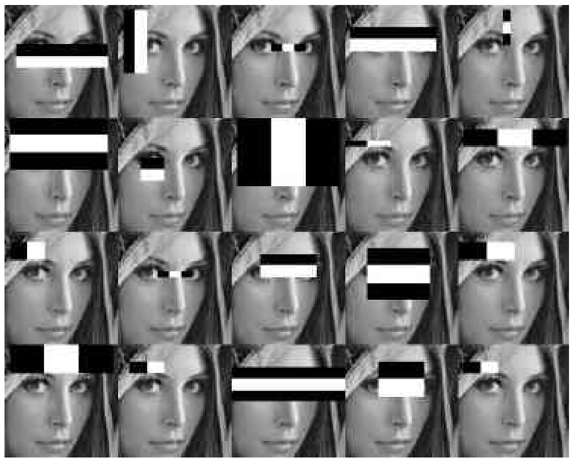

Figure 2: Viola Jones can be used for facial detection. Eyes, noses, and foreheads can be identified by a series of Haar-like Features.

Upon conclusion of training, the detector will determine a set of features able to differentiate the positive and negative training sets. Since training is a one-time process and cascades for detected faces have been published in libraries like OpenCV, I will use OpenCV's provided data and instead focus my work on parallelizing the image processing across the resources of a GPU.

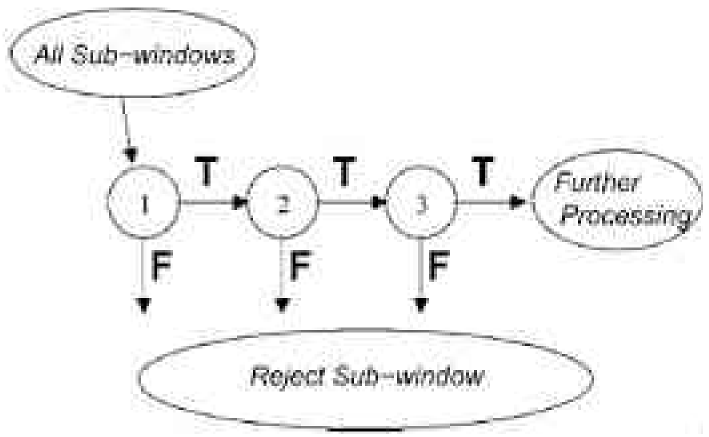

Figure 3: A cascade classifier containing three stages. Each stage will search regions of an image for one or more features. If the region does not contain appropriate feature responses, the possibility of the region containing the target object is rejected

The image to be processed is then divided up into regions. Each region of interest will be tested by the cascade classifier. If the region of interest contains the features to pass through the classifier, the Viola Jones Detector will draw a box around the region of interest to specify an area containing the target object. The following psudocode gives a high level overview of the process:

```
For each region in the image:
        For each stage in classifier
                For each feature in the stage
                        Generate response
                        Accumulate response
                If response < threshold
                        Return Fail
        Return Pass
```

The challenge

The primary challenge will be intellingently distributing processing workload among threads in order to keep threads busy. By it's nature, the cascade takes the longest time to process regions of an image that truly do have a face in it, so images with few faces have the potential to throw off the workload balance amongst threads. If all goes well, I will certain explore other modes of parallelism other than just mapping regions of the image to a thread block.

Additionally, I am wary of the pitfalls of trying to parse OpenCV data structures storing the cascade information that I need to process images. Therefore, it may be in my best interest to see if I can get Opencv downloaded to the cluster machines and use its native library functions to do the parsing.

Resources

For my capstone class last semester, 18-545, my group and I implemented a version of the Viola-Jones algorithm on an FPGA, so fortunately I have some experience in the problem space. Additionally, I have come across new materials pertaining specifically to GPU implementations that will guide me.

https://github.com/opencv/opencv

http://cseweb.ucsd.edu/~kastner/papers/fccm10-gpu_face_detection.pdf

https://sites.google.com/site/facedetectionongpu/

Goals & Deliverables

For me, a successful project will be a program that takes an input image and detects faces on it in a hard real time environment. OpenCV's current implementation runs at about 2-3 FPS, with a GPU, research suggests that processing an image at 30 FPS is an achievable goal.

If all goes well, and I am able to process stored images at 30 FPS, I have a couple stretch goals in mind. First, I will explore other avenues of parallelism in the algorithm, as discussed earlier. Second, and perhaps cooler, I would love to get a camera hooked up to the system and run a video stream that shows faces being detected in real time. The detection would be indicated by a box around the face, and it would follow you around as long as you remain in the camera's view.

Platform Choice

I will be using an NVIDIA GeForce 1080 GTX and CUDA and C++ code to implement this project, as speed is of the essence.

Schedule

Week 1

- Understand OpenCV's parsing of cascades and write a simple program that demonstrates proper parsing of the cascade data structure.
- Begin building necessary framework to facilitate testing
    - Gather test images
    - Build non-processing parts of the program, such as sections dealing with I/O, and any further cascade processing

Week 2

- Finish testing framework
- Begin developing kernels to process image

Week 3

- Finish image processing kernels
- Possibly begin to explore other avenues of parallelism

Week 4

- Finish implementing any extra features

Week 5

- Murphy's law recovery period