Real-Time Fire Detection Using CUDA

Wissam Antoun wfa07@aub.edu.lb, Manal Jalloul, PhD mj37@aub.edu.lb

Electrical and Computer Engineering Department, American University of Beirut

Introduction

With the continuing trends of growing cities and urban areas, fire hazards are expected to rise as well, causing more and more damage. To this day, fire detection systems still rely on hardware sensors, these sensors are usually slow since they rely on smoke making contact with the sensor before the fire is detected [1]. While other sensors rely on temperature information, which is inefficient, as most of the damage will have already been done. In addition, such systems are not practical when implemented in large areas such as forests or highways [2]. Increased urbanization has also caused a rise in video surveillance and monitoring [3], which have the potential to be used in early fire detection. The approach suggested in this paper uses the already existing security camera infrastructure to detect smoke at high frame rate with a reduction in latency. GPUs (Graphics Processing Units) have recently become popular in parallel computing platforms, with CUDA (Compute Unified Device Architecture), created by NVIDIA, providing a programming framework to enable the acceleration of compute-intensive applications using NVIDIA GPUs. Multiple high resolution video streams would generate too much data that CPUs can't handle, hence the use of GPUs. The algorithm will rely on color analysis and other image processing techniques to identify and locate the fire. Another possible use of this algorithm is in processing of video streams originating from flying swarm of drones, that can be programmed to monitor large forests, since the performance and FPS (Frames per Second) should be high enough to allow for real time detection and tracking.

Algorithm

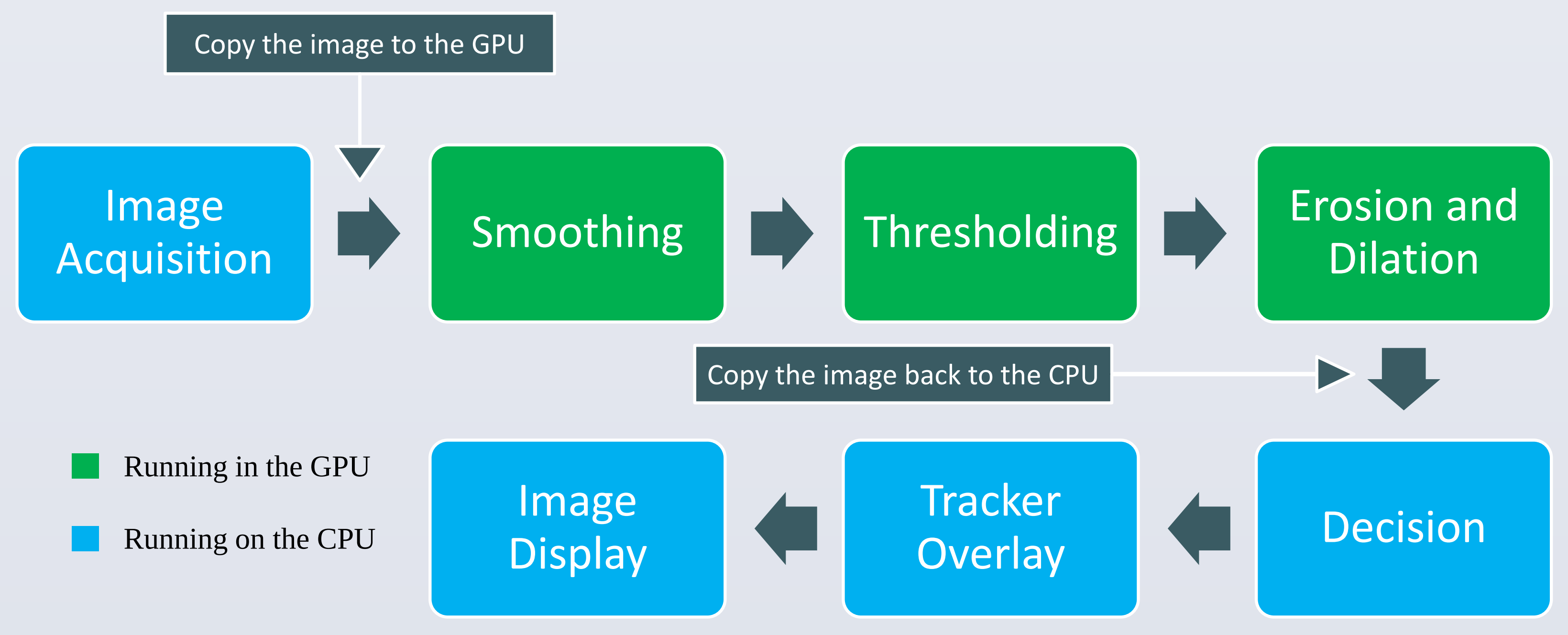

Figure 1. Visual representation of the algorithm

- **Image Acquisition:** To simulate an event where multiple video streams are processed at the same time, the input stream will be a 4K (3840x2176p 14.4Mbps@29.97fps). Which is equivalent to processing 4 FHD video streams or 9 HD streams at the same time.
- **Smoothing:** A smooth filter or a low pass filter is the basis for most smoothing methods. An image is smoothed by decreasing the disparity between pixel values by averaging nearby pixels.
    - CPU: cv::meanBlur() function applies an averaging filter to the image frame
    - GPU: TiledMeanBlur() written using CUDA C, each thread will load the needed pixels (Output pixels + Halo), and only threads within the Output tile will be computing the mean using a Mask that was copied earlier in the initialization phase into the constant memory to guarantee faster access.
- **Thresholding:** checks if the pixel lies between a lower and upper bound
    - CPU: cv:: inRange() function will output a new array that contains only 1 channel with 255 where the original pixel passes the check and 0 otherwise.
    - GPU: GPU_inrange() written using CUDA C, each thread will check if the one pixel is in the range of the supplied arrays.
- **Erosion and Dilation:** Compute a local minimum or maximum (respectively) over the area of the kernel, so in case of Erosion bright areas of the image will get thinner thus removing small noise dots (Fig 2) and in the case Dilation bright areas of the image will get bigger thus overlapping and join objects that are close to each other as shown in Fig 3.

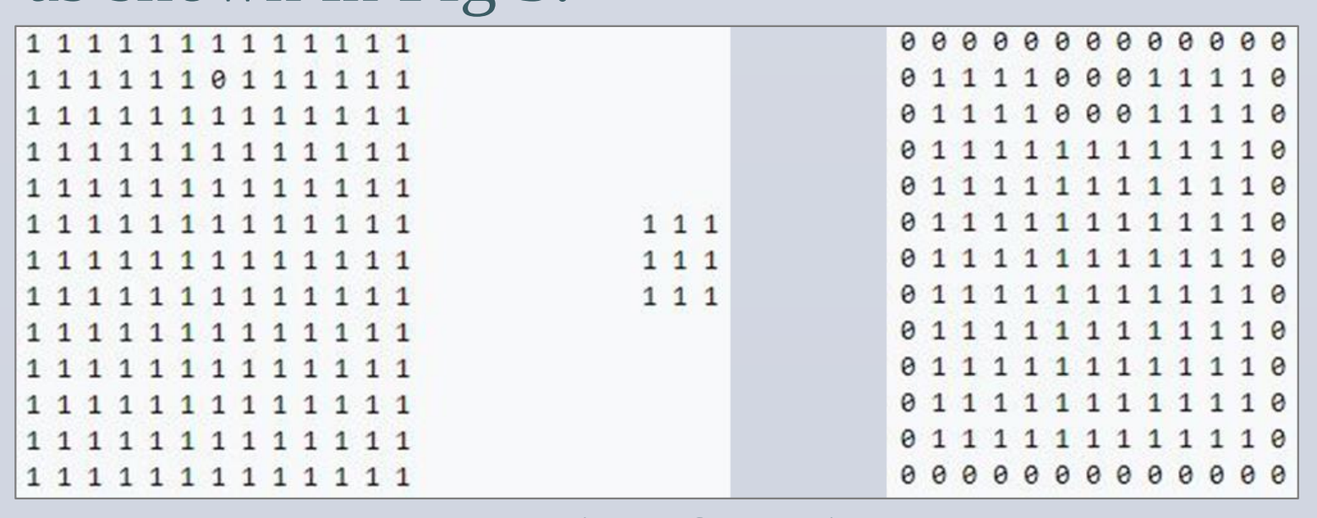
Figure 2. Erosion

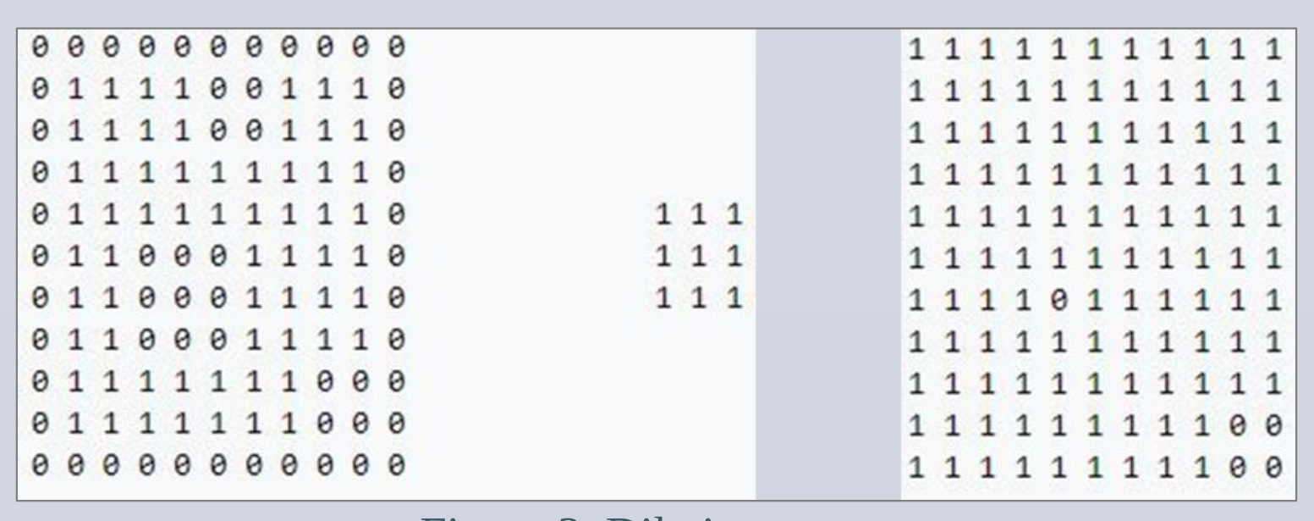
Figure 3. Dilation

- CPU: cv::Erode(), the erode function is used twice in a row to provide better noise removal.
- GPU: TiledGpuErode(), written using CUDA C, each thread will load one of the needed pixels (Output pixels + Halo), and only threads within the Output tile will be checking if one of the value under the Kernel is zero then the central element will replaced by a zero else it will be replaced by 255.

Algorithm (continued)

Decision: The decision is based on the minimum size of the object found in the Threshold matrix after applying the cv::findcontours() function that stores all objects in the matrix in a vector. Also if the function outputs more than 50 objects then the filter isn't accurate and will need adjustments to boundary arrays or the filter masks. Detected Objects will have their coordinates stored in a Vector that will later be used to overlay the tracker on the original image

Experimental Setup

Kernel Parameters :

- BLUR Mask Width 5 Pixels
- DILATE Mask Width 9 Pixels
- ERODE Mask Width 3 Pixels
- 16x16 Threads Per Block
- BLUR GRID SIZE 320x180 Blocks
- DILATE GRID SIZE 480x270 Blocks
- ERODE GRID SIZE 275x155 Blocks
- Threshold GRID SIZE 240x135 Blocks

Computer Specs:
- Intel® Core™ i5-6600 Processor (6M Cache, up to 3.90 GHz)
- 8GB DDR4 2166MHz RAM
- 5600RPM 1TB HDD
- GTX 970 1050 MHz 4GB GDDR5 1664 CUDA Cores

Threshold Boundaries:
- BLUE_MIN = 40;
- GREEN_MIN = 0;
- RED_MIN = 95;

- BLUE_MAX = 256;
- GREEN_MAX = 95;
- RED_MAX = 256;

Results

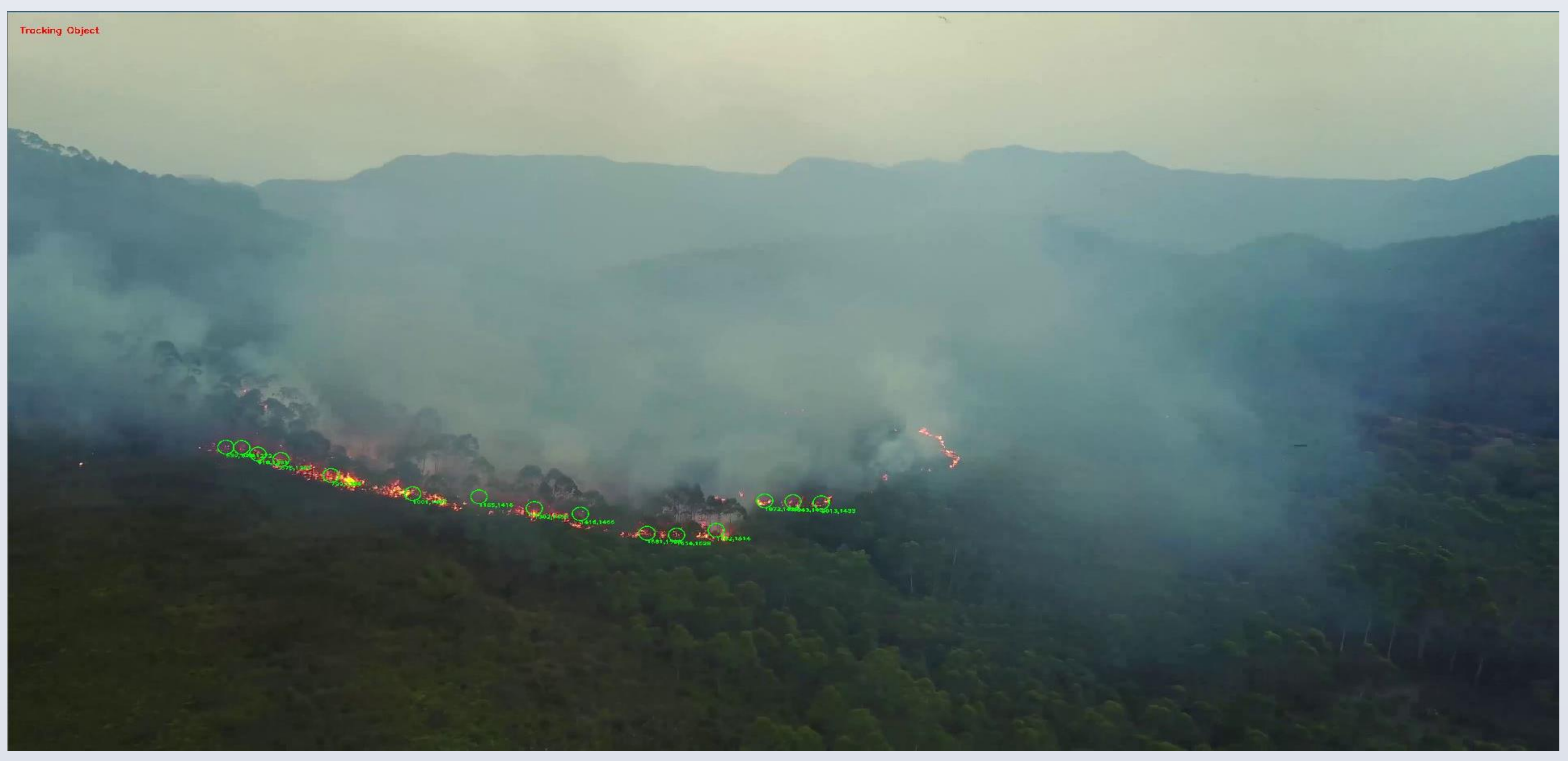

Figure 4. A frame with the tracking overlay

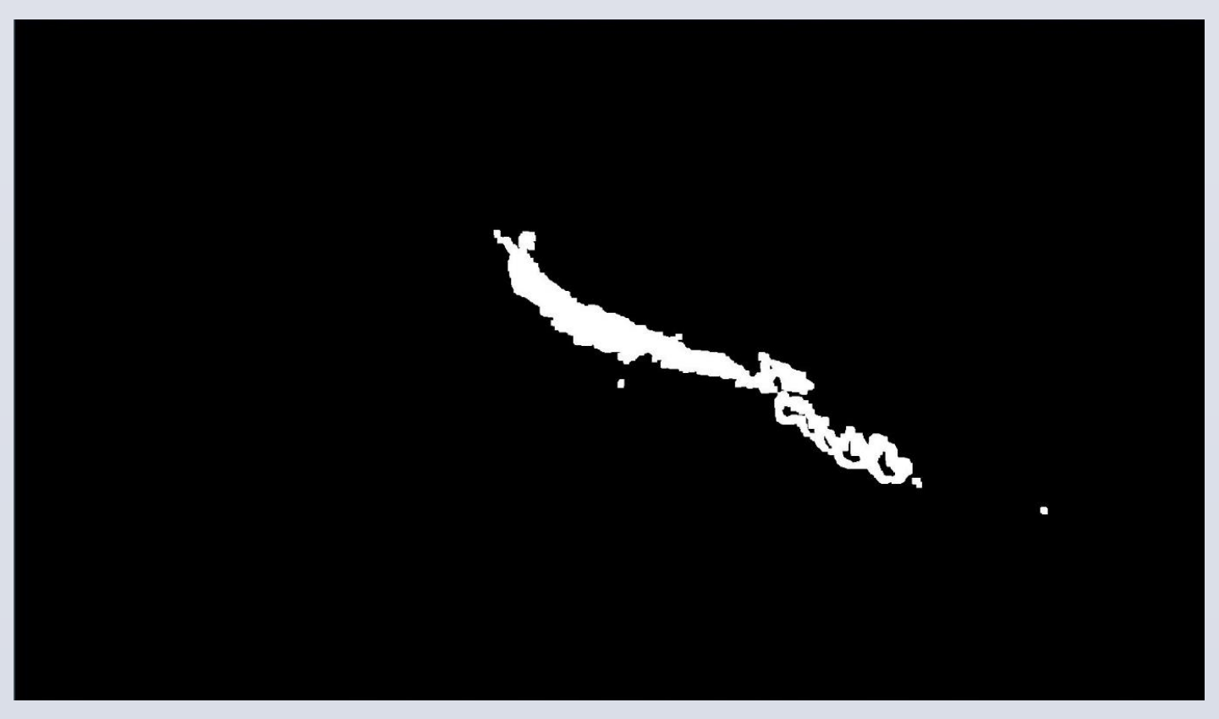
Figure 5. Shows a Threshold frame after dilate and erode

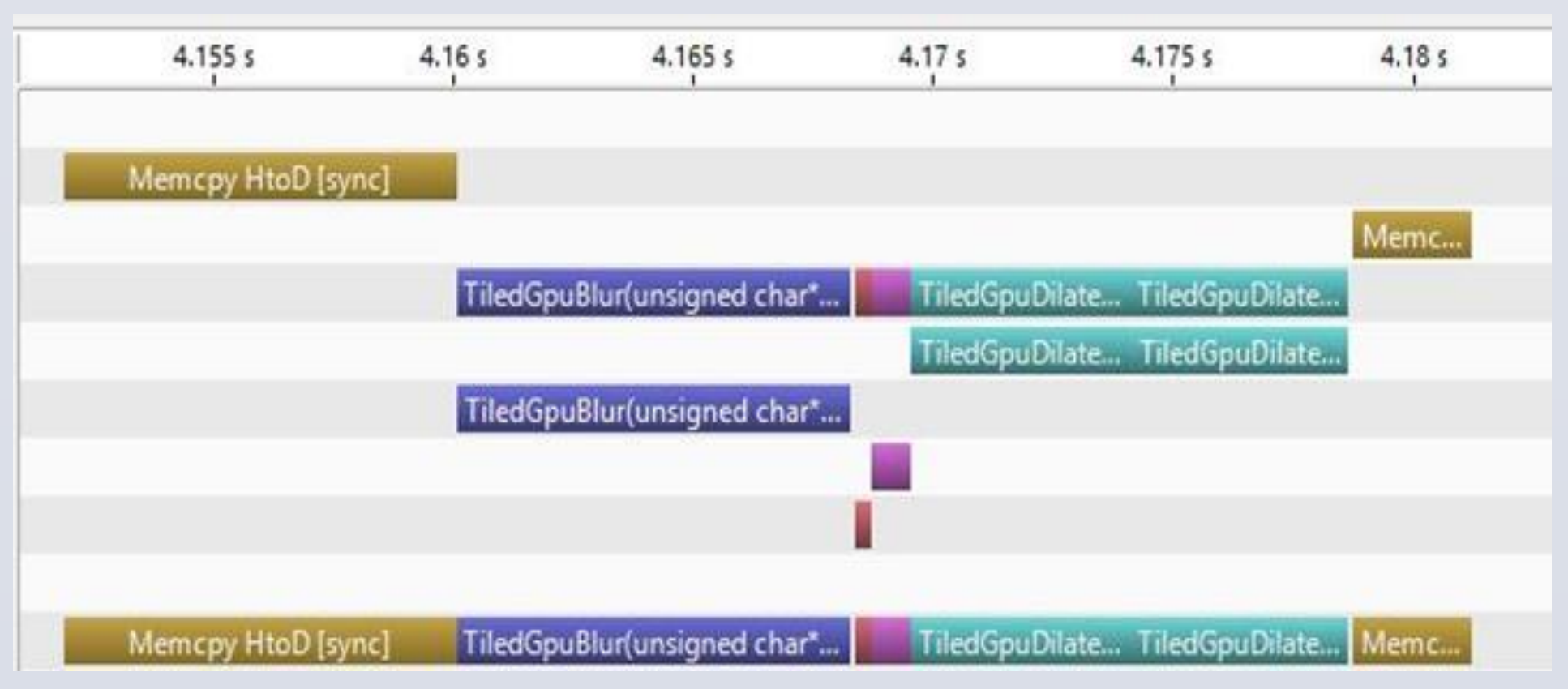

Figure 6. NVIDIA Visual Profiler Output Graph

Table 1. FPS Comparison between different parts of the algorithm

	CPU	GPU	SpeedUP
FPS (4K)	6	11	83%

Conclusion

This research presents a efficient fire detection algorithm that uses image processing to segment fire regions. The algorithm was accelerated by parallelizing using CUDA and implemented on Nvidia's GPU. A speedup of 83% was achieved as compared to the serial implementation. Future Work will explore using CUDA streams to achieve real-time processing.

References

[1] Spearpoint M J, Smithies J N. Practical comparison of domestic smoke alarm sensitivity standards. Proc. 11th International Conference on Fire Detection, AUBE '99, pp. 576- 587, Duisburg, Germany, 1999

[2] Azuma T., Gunki S., Ichikawa A.,Yokota M. Effectiveness of a flame-sensing-type fire detector in a large tunnel. Bedford : ITC. 2005

[3] Koskela H., 'The gaze without eyes': video-surveillance and the changing nature of urban space. Department of Geography, University of Helsinki,Finland. 2000

CUDA code can be found in this Link:
https://github.com/WissamAntoun/Cuda-Kernels/tree/master/Fire%20Detection%20CUDA