GAV - GPU Accelerated Vision

Deepali Shinde[1], Mithilesh Said[2], Pratik Shetty[3], Prof. Swapnil Gharat[4]
Computer Engineering Dept., Rajiv Gandhi Institute of Technology,
University of Mumbai,
Mumbai, India.

[1]deepalis0@gmail.com,[2] s.mit7816@gmail.com,[3] shetty.pratik@outlook.com,[4] swapnil.gharat@mctrgit.ac.in

***Abstract*—Visually impaired people find it difficult to navigate around in unknown environments. Obstacles pose a serious threat to their safety. Several solutions have been proposed for this problem, many being Computer Vision (CV) based. However, CV applications are computationally intensive. When implementing real-time computer vision applications on mobile devices like cell phones or laptops, there are additional constraints like processing time, energy consumption, hardware cost and accuracy of response to take into consideration. Existing approaches where the execution takes place serially on the CPU do not satisfy most of these constraints. The idea behind GAV is to get the best of both worlds - throughput and latency oriented tasks by using the Graphics Processing Unit (GPU) along with the CPU in order to improve the performance of CV. GAV is an application intended to run on mobile devices, that uses the camera to capture live video feed and CPU-GPU pair for computation, in order to help visually impaired people navigate through lesser known spaces.**

***Keywords*—Computer Vision, Image Processing, Machine Vision, Parallel Processing, Real-Time Systems.**

I. INTRODUCTION

People battling complete blindness or visual impairments often have a difficult time navigating by themselves in unfamiliar spaces. Physical movement is one of the biggest challenges for people with an impaired vision. This difficulty increases greatly when there are obstacles in their path. Moving around in such environments can be dangerous and it is important to detect the obstacles and notify the person accordingly. GAV – GPU Accelerated Vision – is a real-time machine vision system that can be ported to mobile devices. The main motive behind using mobile devices/smartphones is that they are portable and hassle-free. Their reducing costs make them an attractive, pragmatic and multi-purpose investment. Nowadays, most smartphones come with graphical processing units (GPUs). Even people with visual impairments can operate them with ease.

Following are the main objectives of the application:-

- Detecting obstacles
- Issue voice alert on detecting obstacles
- Give directions to the user in order to avoid the obstacle
- Operate in real-time

The current implementation of our application uses MATLAB along with its CUDA C based Parallel Computing Toolbox to harness the performance of an NVIDIA GPU and thus reducing response time over the more traditional CPU.

II. RELATED WORK

Over the years, extensive research has been done in the field of computer vision. Technology i.e. software and hardware has been developed with respect to navigation for autonomous robots [4][5][6] as well as for the visually impaired community[11]. Out of these, many have used SONAR [13], LIDAR [10], LASER strippers [1][7] and other expensive sensors along with cameras specifically for obstacle and negative obstacle detection. Systems have even been designed for navigation in known environments using color markers, OCR [3] and GPS [2] in unknown environments. The wide array of sensors used has eventually made the system too bulky and impractical to be used by VI people. Many developers have implemented obstacle detection systems that utilize on-board stereo vision cameras like in [9] for better perception. This has been done with decent success, the only problem being that CV with stereo vision requires a high degree of computation and expensive hardware.

Prototypes like [12] use HSI intensity values and histogram matching to detect obstacles from a wheelchair. Very few applications exist that use GPUs for obstacle detection [14] using Optical Flow [15] and Support Vector Machine algorithms.

III. GPU FOR CV APPLICATIONS

Computer Vision applications require the processing of sequences of images and extraction of important information from those images. Processing of images is a task that involves data-parallel processing. Most of the processing is done on a per pixel basis and there is no interpixel dependency. Traditional applications are built to process data serially and hence they are slow when processing large datasets like images or videos. GPUs, on the other hand, are built to process data in parallel using large number of cores

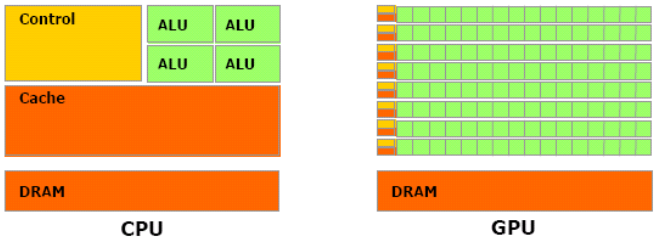

Fig. 1. CPU and GPU architectures

which can deploy several threads that are independent of each other. GPU based computation is well suited for processing arrays, matrices and images. Given below is an architectural comparison of a CPU and a GPU.

CPUs are designed for latency oriented tasks where data accesses are random and there is interdependency between the data elements. They house a large cache and a large control unit. GPUs have smaller control units but a large number of processing cores. Their SPMD (Same Program Multiple Data) architecture make more suited for throughput oriented tasks like computer vision. The use of GPUs for performing intensive and repetitive computational tasks on the huge datasets leaves the CPU free for other high level tasks. It is like using the GPU as a high performance coprocessor for reducing the load on a CPU. CV applications may be safety critical [16] i.e. instantaneous results may be needed. Real-time navigation for the visually impaired falls in that category. The sooner the user gets a warning about an impending obstacle ahead, the more in control he/she is. This inspired us to develop GAV.

IV. System Overview

A. *Requirements*

GAV is at a prototype stage and functions on a laptop PC. However, its eventual use is as an application for smartphones that house chipsets with an inbuilt CPU and GPU. CV based applications are known to be compute intensive, in-turn power sap-ping. GPUs consist of temporally less efficient smaller cores as compared to CPUs but are much more in number. Hence GPU cores are more power efficient. This is advantageous to our application since today's smartphones have a limited battery capacity.

The application requires minimal processors and sensors that are present in most modern day smartphones. These are the CPU, GPU and a camera.

B. *Application setup*

The mobile phone is mounted on the person such that the camera on the device faces in front, away from the user. This ensures that the line of sight and direction is the same for the user as well as the camera. The camera tracks obstacles from ground up approximately up to the user's height. The range for detecting medium to large obstacles in the path of the VI person is between 2 to 2.5 metres. When an obstacle is detected, the user is notified using voice synthesis. For the audio feedback, we use operating system speech facilities, which are standard in modern portable computer systems and smartphones. We currently use the Microsoft Speech software development kit (SDK), which conveniently supports imported script files. [3]

C. *Process Flow*

1. The camera on the device is used to acquire live video feed of 352x288 pixel resolution.
2. The CPU converts the video to frames and accesses frame for processing.
3. Memory is allocated for the frame on the GPU. A captured frame is transferred to the GPU and is processed in a CV processing pipeline comprising of several instructions which perform the processing on the GPU and return the result back to the CPU.
4. The traversability measure is decided based on how big the obstacle is and whether it lies it the path of the VI person. The person is warned if an obstacle is detected ahead of him/her.

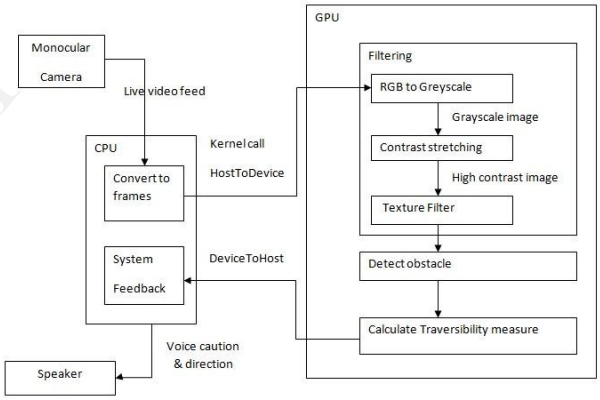

Fig. 2. Process Flow with GPU pipeline

D. Decision Logic

Seen below are snapshots of GAV in use which represent scenarios outdoors and indoors.

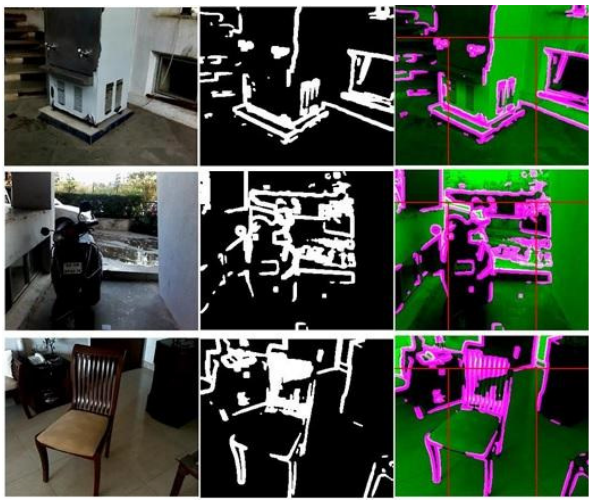

Fig. 3. Rows 1- 3 represent 3 different obstacle scenarios with GAV - a water cooler, a two-wheeler and a chair. Columns 1 - 3 show the (a) Original image (b) Filtered image (c) Fused image respectively

The images in the third column are fused images of the original and the filtered image. The grid seen in those images is used as a guideline for computation and providing directions to the VI person.

Each of the bottom three grids - the left, centre and right - has a threshold, namely *threshLeft*, *threshCentre* and *threshRight*. When the 'white' pixels in the filtered image exceed the threshold for the grid, it means an obstacle is present in that direction. So depending on the conditions satisfied, the user is either told to 'turn left', 'turn right' or that it is a 'dead end'. We refer to the 'white' pixels in each grid as *l(white)*,*c(white)* and *r(white)*.

The four cases are as follows:-
1. When *c(white)<threshCentre*, no obstacle detected.
2. When *c(white)>threshCentre and l(white)<r(white)*, 'obstacle detected', 'turn left'.
3. When *c(white)>threshCentre and l(white)>r(white)*, 'obstacle detected', 'turn right'.
4. When *c(white)>threshCentre and l(white)>threshLeft and r(white)>threshRight*, 'obstacle detected', 'dead end'.

All feedback to the user is given using voice output.

V. Performance

Images of various resolutions were given to the system as inputs to test the speedups achieved by using the GPU over the traditional CPU method. Given below is a performance comparison of the processing times taken by the CPU and that taken by the GPU to perform RGB to GREY function on the images. We consider the RGB to Grayscale since it is one of the tasks performed in our application. Same set of images were used for both the implementations. In most cases the GPU implementation of the code gave 3x speedup on an average as compared to the CPU version. GPUs are built to process large arrays and matrices in parallel. As a result, the larger the image size better is the speedup.

Fig. 4. Input test images to measure CPU and GPU performance for rgb2gray function - from right to left (a) 4444x3136 (b) 4288x2844 (c) 3508x2479 (d) 2000x1300 (e) 2560x1600 (f) 1600x1200 (g) 960x600 (h) 640x360 (i) 480x300

TABLE I
COMPUTATION TIME FOR RGB2GRAY ON THE 9 IMAGES OF VARYING RESOLUTIONS (VALUES IN MILL SEC) ON DIFFERENT PROCESSORS

Image size	1st Gen i3(CPU)	1st Gen i5(CPU)	2nd Gen i3(CPU)	NVIDIA 620M(GPU)
4444 x 3136	113.15	100.86	106.00	39.30
4288 x 2844	91.73	87.16	87.40	34.80
3508 x 2479	92.74	70.16	72.00	24.60
2000 x 1300	18.16	20.43	17.10	8.70
2560 x 1600	28.02	26.50	27.00	14.00
1600 x1200	13.72	14.51	12.70	6.40
960 x 600	4.10	5.83	3.80	2.30
640 x 360	3.37	6.17	3.00	1.10
480 x 300	2.47	1.99	1.90	0.70

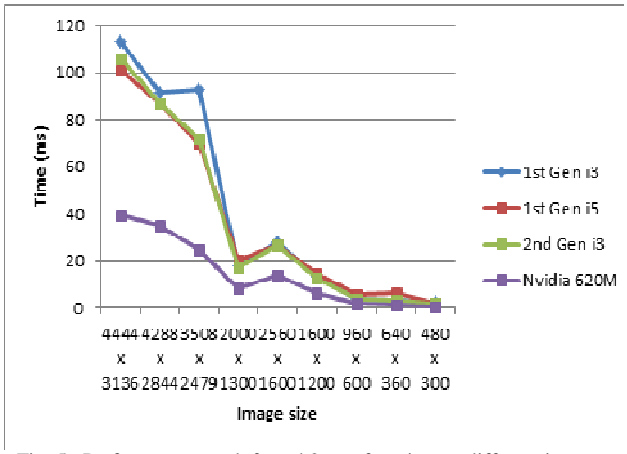

Fig. 5. Performance graph for rgb2gray function on different images

When processing images on a GPU, there are several constraints which decide the performance of an application. Transfer of image data from the host memory to the device memory over the system bus acts as a bottleneck. Larger datasets require a greater memory transfer time. As a result the application needed to be tuned in order to improve the 'compute time to memory transfer time (CMT) ratio. [17]

The traditional CPU implementation of our application produced a frame rate of 9 fps, comparable to other implementations that have ranged between 2.5 to 9 fps. GAV produces a frame rate of 15 fps on an average, thus proving

that GPUs are a perfect fit for CV applications needing real-time results.

VI. CONCLUSION

Machine vision cannot completely replace human vision or senses. It is only an alternative solution in case of weak vision. GAV is one such system which is intended to provide a low cost, but accurate, solution which can enable visually impaired people to traverse with minimum risk to their safety. GAV does not require special hardware and uses what is available on today's smartphone making it affordable to most as compared to other approaches like [18] and [19]. Its performance in real time is a result of the accelerated computations achieved from the on-board GPU. Parallelism in hardware has become inevitable. With increasing computational requirements, it urges us to create better ways for computer vision applications to reap the benefits of parallel computing and create tools to make their lives easier. Given the preponderance of numerical computations in computer vision, it is crucial to focus on ways to improve their performance on modern parallel hardware. Improved performance would imply better response time and hence even more reliable systems in the future.

VII. FUTURE WORK

Our future research with respect to GAV includes porting the application to the smartphone platform and utilizing hardware present on them most efficiently. The CV pipeline on the GPU would be optimized further to achieve a higher speedup. The application must cater to the needs of the user and be easy to operate. Sensors would be used to check orientation and safety of the user. Voice commands and gesture recognition would be implemented to give easy access to the application settings.

REFERENCES

[1] J. D. Yuan and R. Manduchi, "Dynamic Environment Exploration Using a Virtual White Cane", *CVPR*, 2005

[2] Ian Lenz, Mevlana Gemici, Ashutosh Saxena, "Low-Power Parallel Algorithms for Single Image based Obstacle Avoidance in Aerial Robots", *IROS*, 2012.
Available:http://www.cs.cornell.edu/~asaxena/papers/lowpower_obstacleavoidance_mav.pdf

[3] YingLi Tian, Xiaodong Yang, Chucai Yi, Aries Arditi, "Toward a computer vision-based wayfinding aid for blind persons to access unfamiliar indoor environments", *Machine Vision and Applications*, April 2013, Volume 24, Issue 3, pp 521-535. Available: http://media-lab.engr.ccny.cuny.edu/Paper/2012/NavigationAid_CameraReady.pdf

[4] E. Morimoto, M. Suguri and M. Umeda, "Obstacle Avoidance System for Autonomous Transportation Vehicle based on Image Processing", *CIGR*, E-Journal Volume 4, Dec 2002.
Available:http://dspace.library.cornell.edu/bitstream/1813/10295/1/PM%2001%20009%20Morimoto.pdf

[5] Parag H. Batavia and Sanjiv Singh, "Obstacle Detection Using Adaptive Color Segmentation and Color Stereo Homography", *Proceedings of the IEEE International Conference on Robotics and Automation*, May, 2001.
Available: http://www.rec.ri.cmu.edu/projects/toro/tech/color_seg.pdf

[6] Parag H. Batavia and Sanjiv Singh .,"Obstacle Detection in Smooth High Curvature Terrain. Robotics and Automation", *Proceedings ICRA* 2002.
Available:http://repository.cmu.edu/cgi/viewcontent.cgi?article=1995&context=robotics

[7] Romuald Aufrere, Christoph Mertz, and Chuck Thorpe, "Multiple Sensor Fusion for Detecting Location of Curbs, Walls, and Barriers", *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, June, 2003.
Available:http://www.ri.cmu.edu/pub_files/pub3/aufrere_romuald_2003_1/aufrere_romuald_2003_1.pdf

[8] Jason Campbell, Rahul Sukthankar, Illah Nourbakhsh, Aroon Pahwa, "A Robust Visual Odometry and Precipice Detection System Using Consumer-grade Monocular Vision", *Robotics and Automation, ICRA* 2005.Available: http://www.cs.cmu.edu/~rahuls/pub/icra2005-rahuls.pdf

[9] Arturo Rankin, Andres Huertas, and Larry Matthies, "Evaluation of Stereo Vision Obstacle Detection Algorithms for Off-Road Autonomous Navigation", *AUVSI Symposium on Unmanned Systems*, 2005. Available:http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/38362/1/05-1610.pdf

[10] Jacoby Larson and Mohan Trivedi, "Lidar Based Off-road Negative Obstacle Detection and Analysis", *Intelligent Transportation Systems (ITSC)*,2011.
Available: http://cvrr.ucsd.edu/publications/2011/Larson_ITSC2011.pdf

[11] Roberto Manduchi, James Coughlan**,** "(Computer) Vision Without Sight", *Communications of the ACM*, Vol. 55 No. 1, Pages 96-104. Available:http://cacm.acm.org/magazines/2012/1/144819-computer-vision-without-sight/fulltext

[12] Iwan Ulrich and Illah Nourbakhsh, "Appearance-Based Obstacle Detection with Monocular Color Vision", *Proceedings of the AAAI National Conference on Artificial Intelligence*, Austin, TX, July/August 2000. Available: http://www.cs.cmu.edu/~illah/PAPERS/abod.pdf

[13] J. Borenstein, Member and Y. Koren, "The Vector Field Histogram - Fast Obstacle Avoidance For Mobile Robots", *IEEE Journal of Robotics and Automation*, Vol 7, No 3, June 1991, pp. 278-288. Available: http://www-personal.umich.edu/~johannb/Papers/paper16.pdf

[14] K.B. Kaldestad, G. Hovland, D.A. Anisi, "3D Sensor-Based Obstacle Detection Comparing Octrees and Point clouds Using CUDA", *Modeling, Identification and Control*, Vol. 33, No. 4, 2012, pp. 123-130, ISSN 1890-1328.Available: http://www.mic-journal.no/PDF/2012/MIC-2012-4-1.pdf

[15] S. Jeevith, N.D. Gangadhar, S. Santhosh Kumar, "Design And Development Of Optical Flow Based Obstacle Avoidance Using CUDA", *SAS Tech – Journal*, Vol 10, Issue 2, 2011. Available: http://www.msrsas.org/docs/sastech_journals/archives/sept2011/3.pdf

[16] Ron Tal, "Real-Time Approaches to Computer Vision", 21 May 2009.

[17] Deepali Shinde, Mithilesh Said, Pratik Shetty, Swapnil Gharat, "Optimizing Real Time GPU Kernels Using Fuzzy Inference System", *IJARSE*,Vol.No.2,Issue No.9, September 2013.
Available:http://www.ijarse.com/images/full_pdf/1380117678_OPTIMIZING_REAL_TIME_GPU_KERNELS_USING_FUZZY_INFERENCE_SYSTEM.pdf

[18] Juan Manuel Saez Martinez, Francisco Escolano Ruiz, "Stereo-based Aerial Obstacle Detection for the Visually Impaired", *Workshop on Computer Vision Applications for the Visually Impaired* (2008).
Available: http://hal.inria.fr/docs/00/32/54/55/PDF/AerialObstacles.pdf

[19] Mobile Vision Research Lab, "Aerial obstacle detection software for the visually impaired", Universidad de Alicante