**Edge Detection using Sobel and Canny filter on CUDA**

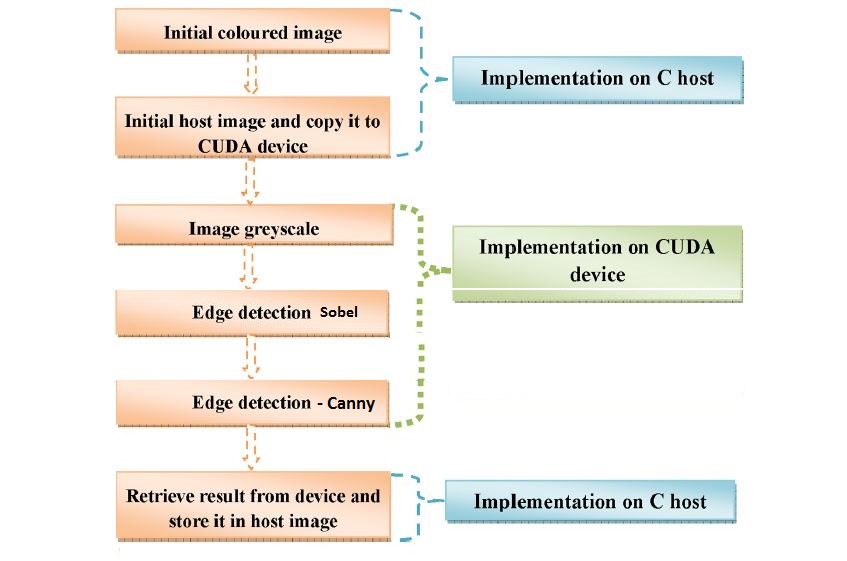
Mallepalli Rakesh Reddy

Ramidi Tarun Reddy

**Introduction:** Edge detection is the name for a set of mathematical methods which aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. It is one of the fundamental steps in image processing, image analysis, image pattern recognition, and computer vision. The assignment problem on image grey scale conversion has given us basic idea on how parallel programming helps in image processing techniques. We have done a project “Object tracking by a robot” in under graduation using image edge detection and MATLAB. We intend to implement the edge detection technique in parallel and demonstrate its speed up using CUDA platform.

**Objectives and Design:**

The aim of our project is to implement the Sobel and Canny edge detection algorithm, as it encompasses a set of image processing techniques.



Parallel Programming Patterns:

Since Canny filter computations can obviously be divided into a number of completely independent parts, each of which can be executed by a separate processor; we make use of SIMD (Single Instruction Multiple Data) pattern so that we can launch multiple parallel threads to perform an operation on each pixel, with this we can achieve high computational speed.

Canny Filter Pseudo code:

A possible algorithm consists of the following steps:

1. **Noise reduction.** May be performed by [Gaussian filter](http://en.wikipedia.org/wiki/Gaussian_blur).
2. Compute **intensity gradient** (matrices *Gx* and *Gy*) and its **magnitude** *G*.  
      G=\sqrt{G_x^2+G_y^2}  
   May be performed by convolution of an image with Sobel operators.
3. **Non-maximum suppression.** For each pixel compute the orientation of intensity gradient vector: \theta = {\rm atan2}\left(G_y, \, G_x\right). Transform angle θ to one of four directions: 0, 45, 90, 135 degrees. Compute new array *N*: if  
      G\left(p_a\right)<G\left(p\right)<G\left(p_b\right)  
   where *p* is the current pixel, *pa* and *pb* are the two neighbor pixels in the direction of gradient, then *N*(*p*) = *G*(*p*), otherwise *N*(*p*) = 0. Nonzero pixels in resulting array correspond to local maxima of *G* in direction θ(*p*).
4. **Tracing edges with hysteresis.** At this stage two thresholds for the values of *G* are introduced: *Tmin* and *Tmax*. Starting from pixels with N(p) \geqslant T_{max} find all paths of pixels with N(p) \geqslant T_{min} and put them to the resulting image.

**Performance Goals and Validation:**

To get maximum speed up by using parallel programming and validate the result by comparing final images from parallel and sequential code.

When the image size increases, time taken for memory access increases. Our goal is to make effective use of cuda device memory to make more coalesced access and get more speed up

**Schedule:**

|  |  |  |
| --- | --- | --- |
| Dates | Person/s | Responsibility |
| 11/16/2015 to 11/18/2015 | Ramidi Tarun Reddy | Convert colored to grey scale. |
| 11/16/2015 to 11/18/2015 | Mallepalli Rakesh Reddy | Apply Gaussian filter. |
| 11/19/2015 to 11/25/2015 | Ramidi Tarun Reddy | Apply Sobel filter. |
| 11/19/2015 to 11/25/2015 | Mallepalli Rakesh Reddy | Apply Canny filter. |
| 11/25/2015 to 11/30/2015 | Rakesh and Tarun | Integration and Optimization |

We will maintain the source code in GitHub.

**References:**

<http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_Chapter5.pdf>

<http://www.nvidia.com/content/nvision2008/tech_presentations/Game_Developer_Track/NVISION08-Image_Processing_and_Video_with_CUDA.pdf>

<http://www.ijraset.com/fileserve.php?FID=76>

<http://www.researchgate.net/publication/261959917_Efficient_implementation_of_Sobel_edge_detection_algorithm_on_CPU_GPU_and_FPGA>