Fast Hough Transform on GPUs: Mid-Term Report

Madhur Tandon (2016053)
Mudit Garg (2016057)

I. Abstract

Feature Extraction is an important step in Image Processing, Computer Vision and other related fields. One of the major features in an image are detection of instances of objects of a certain type. Hough Transform is a technique that helps to identify these objects via a voting mechanism. The objects or shapes are detected as a local maximum in the accumulator space which is implicitly constructed by the underlying algorithm.

Hough Transform has been initially applied to detect lines in an image. Further developments also include the detection of circles and in recent years, the Hough Transform has also been generalized to detect any arbitrary shape.

The algorithm usually involves many steps in the pipeline and thus takes a very long time to get good results. Since detection of these geometrical figures is only one step of a larger problem - say classification, it becomes difficult and quite impossible to be able to carry out real-time image processing pipelines.

With the advent of NVIDIA's GPUs and the CUDA API, efficient computations on matrices and vectors can be carried out exploiting the massively parallel capabilities of the GPU's hardware. In fact, almost every step of the pipeline can be parallelized and significant speedups can be obtained.

II. Analysis of Algorithm

The Hough Transform Algorithm includes several steps in the pipeline. These include

- Conversion To Grayscale
- Blurring of an Image
- Edge Detection
- Creation of Accumulator Array
- Thresholding
- Non-Maxima Suppression

Each of the steps in the pipeline are highly parallelizable. We discuss the above steps below:

A. Conversion To Grayscale

Coloured Images are primarily 3D in nature. There are 3 channels for Red, Green and Blue each. The conversion formula to grayscale is given by

$$0.299 * R + 0.587 * G + 0.114 * B$$

This is a pixelwise operation and this can be applied on each pixel independent of other pixels. Thus, a highly efficient implementation using GPU can be achieved.

B. Blurring of the Image

The images need to be blurred to remove some noises so that only those edges are kept which are relevant. An averaging kernel or a gaussian blurring kernel can be used to perform this operation. Regardless of which kernel is used, this process is essentially a convolution operation and the convolution of a kernel with an image can be performed parallely with the values being computed for each pixel as the kernel's center by a separate thread.

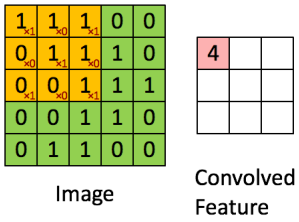

Fig. 1. Convolution Operation

C. Edge Detection

The edges in an image are the points for which there is a sharp change of color. These features can be captured by the gradient of an image. Gradient of Images are calculated using the Sobel Operators. This in fact is another convolution operation just like above.

The magnitude and orientation of the gradient can be computed per pixel independent of other pixels and thus, a GPU based implementation can help speed up this operation.

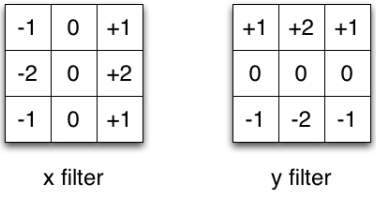

Fig. 2. Sobel Operators

D. Creation of Accumulator Array & Thresholding

1) Algorithm for Circles:

An equation of a circle is given by

$$(x-a)^2 + (y-b)^2 = r^2$$

. But this equation is of little use and the more general parametric equation

$$x = a + r * Cos(Q)$$

and

$$y = b + r * Sin(Q)$$

is used where Q ranges from 0 to 360 degrees.

The edge points "x", "y" in the image domain are put in the above equation for a fixed radius "r" and for "Q" varying from 0 to 360. The corresponding values of "a" and "b" are found out and if these centers lie within the height and width of the image, the accumulator array at the index

$$a, b, r$$

is incremented. (Voting Mechanism)

For the filled accumulator array, only those "a, b, r" are retained which have at least 40% of the total points as their votes. This threshold can be changed according to one's convenience.

2) Algorithm for General Geometric Shapes:

First, The object which needs to be detected needs to be represented in its parametric equations. For this, a reference point (centroid) of the edges of the object is found. Further, the angle made by each edge point with this reference point and the axis is found out. This is done by using the orientation of the sobel operator in x and y direction and We call this angle theta. There exist many points along the edges that have the same angle with the reference point. We call the vector joining the reference point to the edge point as "r". These vectors "r" are stored in a dynamic array indexed by the angles (theta). Finally an accumulator array is created and populated by voting for each "theta, r" pair for all the points lying on the edge of the image.

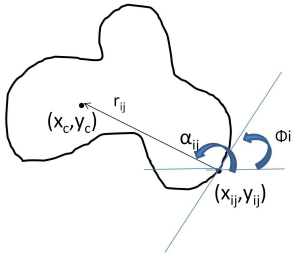

Fig. 3. General Shape Detection

III. Parallelization Strategy and Implementation Notes

Several Kernels are made which perform respective steps of the pipeline. Currently, for the mid-term goal, this has been done specifically for the Circle Detection using Hough Transform. These are mentioned below:

- grayscaleKernel
- convKernel
- sobelKernel
- accumulatorKernel
- thresholdingKernel

These perform the relevant operations and each kernel is run one after another to produce the final output. Out of the above, the kernels "accumulatorKernel" and "thresholdingKernel" are 3D kernels i.e. they use the "z" dimension of *blockIdx*, *blockDim* and *threadIdx* as well. The other kernels are 2D Kernels.

IV. Results of Implementation

The following Images show the results obtained after each step of the pipeline. All these are obtained using the kernels which are created above and are run sequentially. The loading and saving of images is done using the *stb_image.h* and *stb_image_write.h* library which was given to us in the first assignment of doing convolution on GPU.

Fig. 4. Input Image

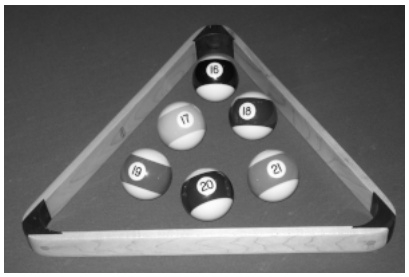

Fig. 5. RGB to Grayscale

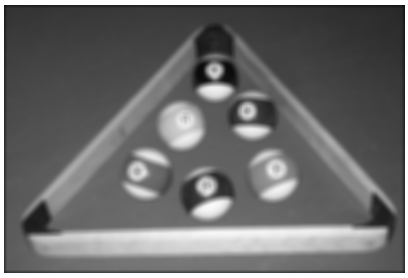

Fig. 6. Blurred using Averaging Kernel

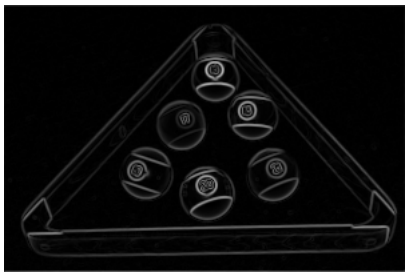

Fig. 7. Edge Detection using Magnitude of Sobel Operator

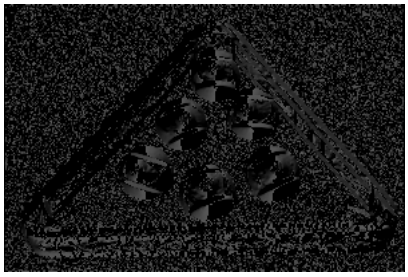

Fig. 8. Orientation of gradient obtained using Sobel Operator

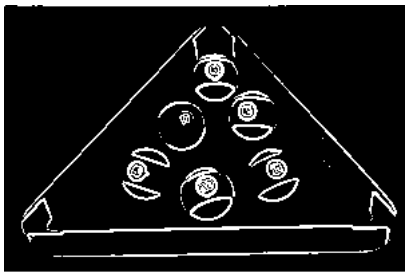

Fig. 9. Thresholding on the detected edges

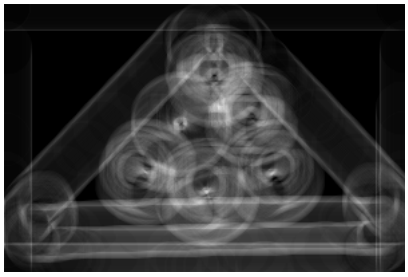

Fig. 10. Hough Space for one of the fixed radius

Fig. 11. Detected Circles plotted using openCV in Python

V. Next Set of Milestones

- Performance Analysis with Serial CPU code
- Serial CPU Code for generic geometric shapes
- Hough Transform GPU Code for generic geometric shapes

References

[1] Su Chen and Hai Jiang
Accelerating the Hough Transform with CUDA on GraphicsProcessing Units