Pavlo Bazilinskyy
Mikkeli University of Applied Sciences, Finland

Implementing Hough transformation with C language of programming

The Hough transformation is a technique, which is used to detect and isolate objects of a predefined shape within an input image. The classical realisation of the Hough transformation requires defining types of shapes to be processed on input. The most commonly used utilisation of this tool is detection of lines or circles in a give image [6]. The Hough transform as it is mostly known today was introduced by Richard Duda and Peter Hart in 1972, they called their technique a "generalized Hough transform" [7].

This article takes a case of finding straight lines in the given image into account. In order to find a line on the image a set of discrete points that come out from such filtering algorithm as edge detection must be plotted into a line segment. The equation for a line is $y = ax + b$ and it can be plotted for each pair of points (x, y) on the image. As stated by McAullife (Year not give), a task of the Hough transformation is to find find the slope parameter a and the intercept parameter b for each line and then output the line. However, the slope approaches infinity as the line becomes more vertical. This issue can be resolved if a parametric equation is used:

*x *cos(theta) + y*sin(theta) = rho*

where, *rho* is the length of a normal from the origin to the line and *theta* is the orientation of the line with respect to the X axis.

After (rho; theta) values are found for each point in the image the Hough space can be drawn. On the received graph points with the biggest concentrations of plotted dots indicate lines in the image.

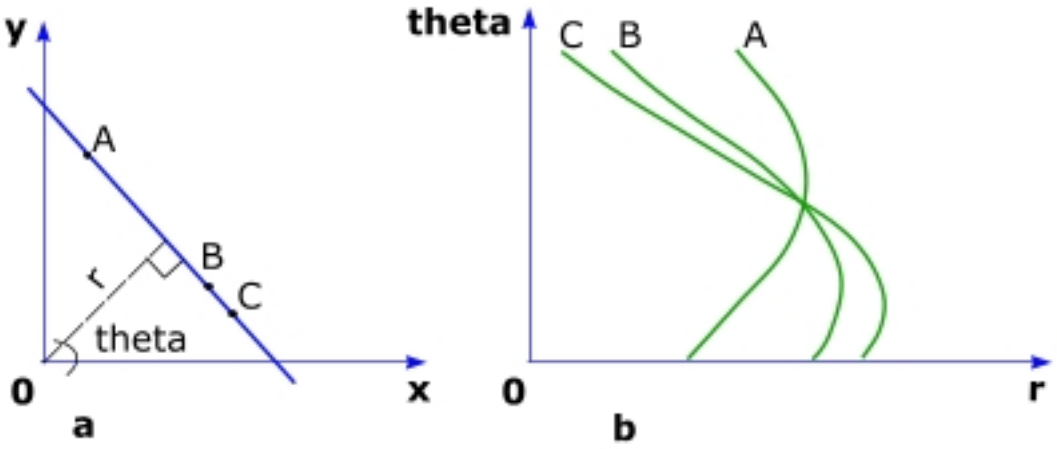

Figure 1. (a) a straight line in the original coordinates described in terms of the length of a normal from the origin to the line - r and orientation theta; (b) the Hough plane where points A, B, and C are transformed into three sinusoidal curves [7].

The Hough transformation can be relatively easily implemented using such common languages of programming as C, C++ or Java [4]. In the scope of this article working with this technique in C language of programming is described. The algorithm that can be used to implement the Hough transformation can be outlined:

1. Populate an array with (rho; theta) values.
2. Detect number of lines.
3. Draw Hough space.
4. Draw lines based on the Hough space.

Firstly, a function for performing detecting lines *houghTransformation()* can be created. Extensive research was done on usage of matrixes with CUDA to store theta/ rho values, but better results were achieved in handling 1d arrays on GPU level [1, 2]. To achieve that the accumulator array should be utilized in which “votes” are given for each point in the image that is given on input. The following approach can be used to populate a 1-dimensional array with (rho; theta) values for each point in the inputted image: votes are put into an accumulator array, where index *i* is calculated: *i = rho + (theta * NUM)*, where *NUM* is a predefined value, that is always larger than a maximum value for *rho*. Hence, *rho = i % NUM* and *theta = i / NUM*.

Then, *drawLine()* function can be written that draws a line for a given theta/rho value. To draw a line on the image, (x;y) coordinates can be calculated using this formula to find *y* for each *x*: *y = (rho - x*cos(theta)) / sin(theta)*.

The next step is to draw the Hough space of dimensions [Max value of rho; 180], where 180 is the max value for theta.

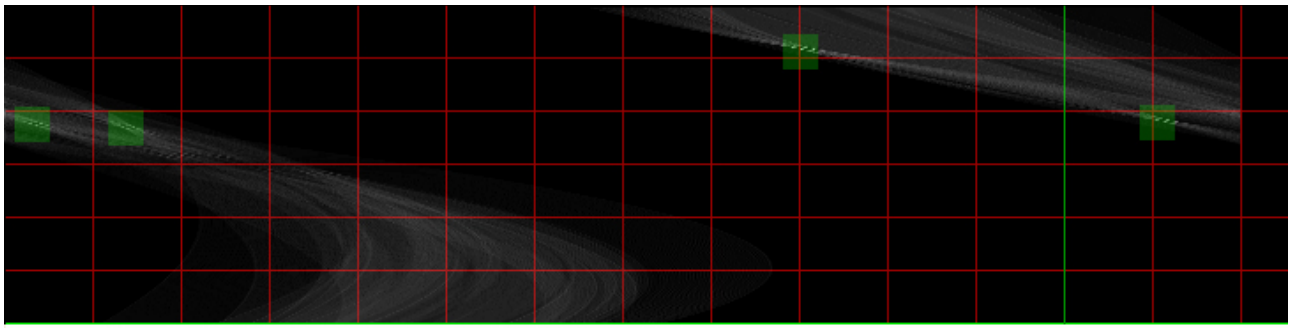

Figure 2. Hough Space with 4 lines detected (brightness increased).

Aforementioned approach works fine for detecting one line object. However, a problem of detecting and drawing multiple lines can also considered. To detect multiple lines a radius around high numbers of votes in the accumulator array, which indicates a detected line, can be defined; then the array can be looped backwards, calling *drawLine()* function for each detected line.

This problems requires complex calculations and thus should be solved using parallel computing. The tool used for writing this article was CUDA library from NVIDIA Corporation. This library allows programmers to utilize C and C++ languages of programming to perform complex parallel calculations [1, 3]. All CUDA functions that are used in examples in this article have <<<width, height>>> dimensions, which are determined on a stage, when an input image is loaded. It allows running *width * height* (dimensions of the image) number of threads to be run simultaneously, which results in satisfying performance.

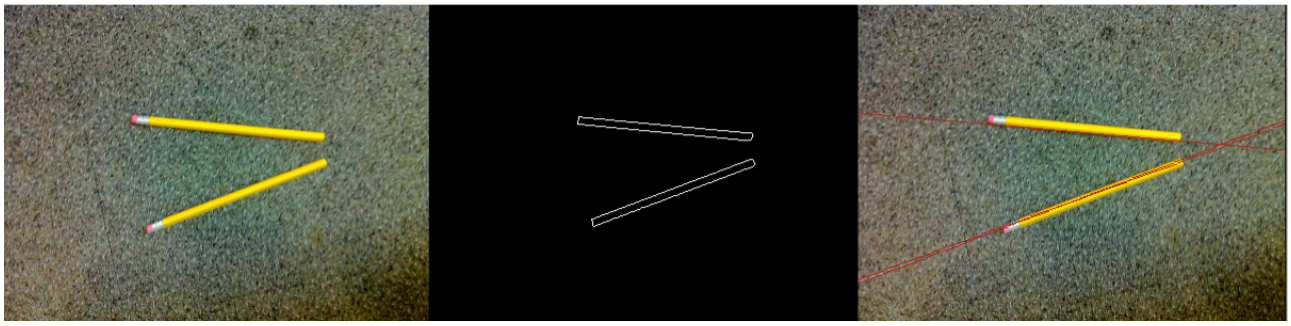

Figure 3. Detection of 2 lines using the Hough transformation.

A final output of the programme is a window with an original image and images, which show transformation that the input image undertakes to come to a final result, that is used in the Hough transformation function. Then, the original image with detected lines drawn on top of it is presented along with the Hough space.

Listings of functions used for the Hough transformation using C language of programming:

houghTransformation():

```
// ============================================================
// CUDA function that calculates rho and theta values for
// detecting lines using Hough transformation algorithm for given values of
// (x;y). It uses an equation rho = x*cos(theta) + y*sin(theta), where x and y are
// coordinates of a pixel on the image. Rho is the length of a normal from the
// origin to  the line, theta is the orientation of the line with respect to the X
// axis. Line detection needs to output results into a 2d space[rho][theta].
// Parameters:      unsigned char *in  - pointer to the array with transformed
//                                       image
//                  int *out           - pointer to the array for storing
//                                        histogram values
//                  int n_pixels       - number of pixels multiplied by
//                                       sizeOf(int)
// Returns:     nothing
```

```
// ================================================
```

*__global__ void houghTransformation(unsigned char *in, int *out, int n_pixels, int width, int height)* {

```
int thread_id = (blockIdx.x * blockDim.x) + (threadIdx.x);
int thetaDeg;
float rho;
if (thread_id < n_pixels) {
        if (in[thread_id] > 0) {
                //Checking the colour. If it is more than zero then it is not black.
                int y = thread_id / width;
                int x = thread_id - (width * y);
                for (thetaDeg = 0; thetaDeg < 180; ++thetaDeg) {
                        //Iterate through theta
                float thetaRad = dev_degrees_to_radians(thetaDeg); // degrees into radians conversion
                 rho = x * cosf(thetaRad) + y * sinf(thetaRad);
                out[rho + (thetaDeg * NUM)]++;//Increment a value in an accumulator array
                }
        }
}
}
```

drawLine()*:*

```
// ================================================
// Function that draws the Hough space using *space.
// Parameters:      int *histogram      - array with theta(Y), rho(X) values.
//                  float rho           - value of rho used.
//                  float theta         - value of theta used.
//                  unsigned char *lineImage        - array to hold generated
```

```
//                                                    image.
//                    int width          - width of the image.
//                    int height         - height of the image.
// Returns:    nothing
// ================================================
```

```
void drawLine(int *histogram, float rho, float theta, unsigned char *lineImage,
int width, int height) {
 int x, y;
 for(x = 0; x < width; x++){
        y = (int) (rho - x * cosf(theta)) / sinf(theta);
        set_pixel(lineImage, x, y, 178, 34, 34, 255, width, height);
 }
}
```

To test the programme images custom images were used along with real photos. These factors affecting results were found: dark lines are harder to detect; solid background is required for better results; if the line is not clear enough the line is detected more than once.

To perform testing different images were used as input and 7 different values for <<<width, height>>> were given, as can be seen from the graphs below along with a control test on CPU level.

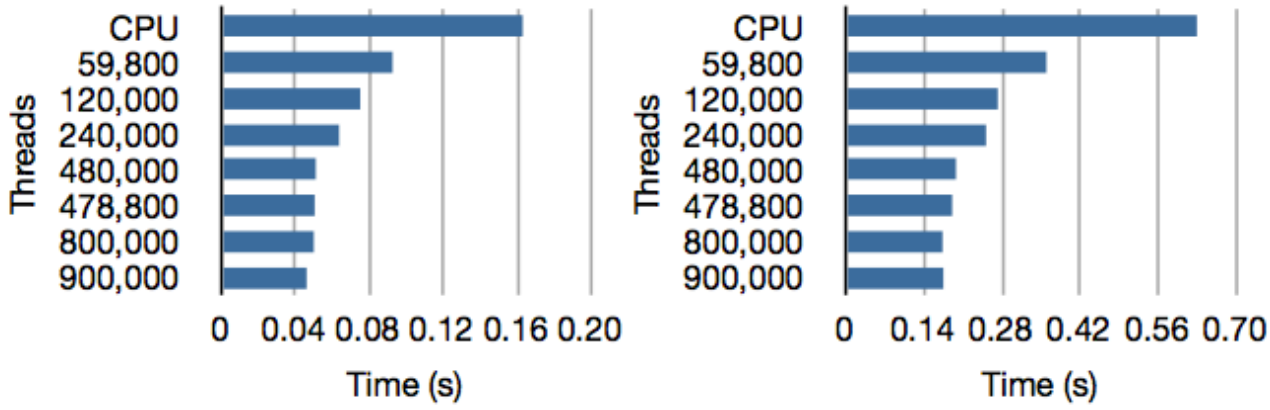

Figure 3. Timing data for images with 1 pencil (left) and 4 pencils (right).

It is clear from testing performed that after certain point increasing number of threads does not affect performance with a linear dependancy. It is, therefore, important to calculate threshold for the optimum number of threads for the best performance and cost efficiency [5].

Literature:

1. **NVIDIA Corporation**. CUDA C Programming Guide. Nvidia Corporation - 2011. Referred 25.1.2011. Available in www-format: <http:// developer.nvidia.com/nvidia-gpu-computing-documentation>.
2. **NVIDIA Corporation**. CUDA C Best Practices Guide. Nvidia Corporation - 2011. Referred 25.1.2011. Available in www-format: <URL: http:// developer.nvidia.com/nvidia-gpu-computing-documentation>.
3. **NVIDIA Corporation**. CUDA C Getting Started Guide for Linux. Nvidia Corporation - 2011. Referred 25.1.2011. Available in www-format: <URL: http://developer.nvidia.com/nvidia-gpu-computing-documentation>.
4. **Kim King**. C Programming: A Modern Approach. 2nd ed. W. W. Norton & Company - 2008.
5. **Stephen Keckler, William Dally, Brucek Khailany, Michael Gerland, David Glasco.** GPUs and the Future of Parallel Computing. Nvidia. Computing Now November, p. 1 - 2011
6. **Matthew McAullife.** Hough Transform, Imaging Sciences Laboratory. Year not given. Referred 26.1.2011. Available in www-format: <URL: http:// mipav.cit.nih.gov/documentation/HTML%20Algorithms/ HoughTransform.html>
7. **Richard Duda and Peter Hart**. Use of the Hough Transformation to Detect Lines and Curves in Pictures. Comm. ACM, Vol. 15, pp. 11–15 - January, 1972