A Fast Line Segment Detector using Approximate Computing

Christoph Ossimitz and Nima TaheriNejad
TU Wien, Vienna, Austria
E-mail: nima.taherinejad@tuwien.ac.at

2021 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-7281-9201-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ISCAS51556.2021.9401660

***Abstract*—The Line Segment Detector (LSD) algorithm is an underlying step of many image processing systems. Hence, its performance has a significant on the upper layers using the detected line segment for various purposed. In this paper, we propose a fast LSD algorithm. This method approximates several floating point operations, including the logarithmic Gamma function, by a series of lookup table searches. Due to the simplicity of such approximation (lookup table search) compared to the naïve implementation (calculation-based), this method is considerably faster. The proposed method has implications on reduction of the necessary efforts to implement and enhancement of the performance of the LSD hardware accelerators. Our experiments show that the proposed method reduces the run-time of the algorithm by 13% on average, with no considerable quality loss in the detection results. This improvement further propagates through other image processing algorithms using LSD.**

I. Introduction

Due to a rising research interest in topics like autonomous driving, there is an increasing demand for real-time computer vision algorithms. Line segment detection algorithms are low-level tasks commonly used in many computer vision applications, such as stereo matching [1], target tracking [2], odometry [3] and detection of barcodes [4], power lines [5], road lanes [6] [7], sea-sky lines [8] and airports [9] [10]. A *line segment* is a locally straight contour on the image, that is, a zone of a sharp change from a light to dark area (or vice-versa) [11]. Line segments are finite and can be characterized by the coordinates of the two end points.

One of the prominent line-segment detection algorithms is the Line Segment Detector (LSD) algorithm published by von Gioi in 2010 [11], featuring the *a contrario* method by Deolneux et. al [12] [13] to validate each line-segment candidate, which allows the algorithm to produce stable detection results, even under the presence of image noise. We have summarized all the processing steps of the algorithm in the following. Internal parameters of the algorithm are introduced with the default or recommended values.

1) Using Gaussian sub-sampling, the input image is scaled down to 80%. This is done to deal with image noise and quantization artifacts (for example, the staircase effect).
2) The image gradient (magnitude and angle) is calculated. If the gradient magnitude of a pixel is below a threshold (denoted by $\rho = \frac{2}{\sin 22.5^\circ}$), the gradient magnitude is set to zero instead.
3) All pixels are binned according to their gradient magnitude into n buckets (default value for n is 1024).
4) Using the pseudo-ordering introduced in the binning, a high-magnitude pixel is selected for region growth. Iteratively, neighboring pixels with a gradient angle within a tolerance (denoted by $\tau = 22.5^\circ$) of the whole region's angle are added to the region.
5) Once none of the neighboring pixels meet the criteria to join the region, a rectangle to encase the region is calculated. If the ratio of region pixels to the number of all pixels inside the rectangle is below a certain threshold (denoted by $D = 0.7$), certain optimization steps are applied to the region/rectangle to increase this ratio. If the threshold D is still not met afterwards, the region is rejected and the algorithm goes back to Step 4, picking a different pixel.
6) The algorithm tries to verify the rectangle as a valid line-segment, by iteratively calculating the Number of False Alarms (NFA) as well as performing certain optimization steps. This is described in more detail below.
7) If the verification is successful, the 2-dimensional rectangle is transformed to a 1-dimensional line-segment. All pixels from the corresponding region are marked as used and can no longer be selected during future iterations of region growth. The algorithm now returns to Step 4, until all pixels are marked as used, at which point the algorithm terminates.

During the line-segment verification step, the NFA for the respective rectangle is calculated. The rectangle is verified once the NFA value is smaller than or equal to a threshold (denoted $\varepsilon = 1$). The NFA value is calculated as

$$\text{NFA}(n,k,p) = (N \cdot M)^{5/2} \cdot \gamma \cdot \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j} \quad (1)$$

where n denotes the number of pixels in the rectangle, p denotes the precision value (initialized with $p = 0.125$, later reduced during the rectangle refinement step). k denotes the number of pixels in the rectangle with gradient angle equal to the rectangle's angle up to a tolerance of $p\pi$. Finally, γ denotes the number of different values used for p (in the default implementation, $\gamma = 11$).

The algorithm recalculates the NFA repeatedly in what is referred to as the *rectangle refinement*, with different values for p as well as decreasing the width of the rectangle. This is done in five different steps, where each step modifies either p or the width of the rectangle up to 5 times each. If none of

978-1-7281-9201-7/21/$31.00 ©2021 IEEE

the calculated NFA values is smaller than or equal to ε, the configuration with the lowest NFA value is kept and carried into the next step. Therefore, per line-segment candidate, the NFA value is calculated up to 26 times. If none of these results is smaller than ε, the candidate is rejected. Otherwise, the candidate is verified. The implementation by Gioi et. al. calculates the decadic logarithm of the NFA value, using approximations by Windschitl and Lanczos, and check this value against $\log_{10}(\varepsilon)$. This is computationally very expensive, and implementing the algorithm in hardware would be either slow or result in a long pipeline with high usage of area.

Hence, instead of calculation the logarithms, we aim to approximate $\log_{10}(\text{NFA}(n,k,p))$ utilizing a series of lookup table entries, decreasing the complexity of the algorithm as well as increasing its speed, while maintaining its accuracy.

II. Proposed Method

1) Requirements: As mentioned above, in this work, we propose to use a lookup table instead of calculating the logarithm values. To keep the size of the lookup table reasonable, a few assumptions have to be made: For one, the resolution of the scaled image $N \cdot M$ has to be known in advance and cannot be changed during operation. In most applications, the input data uses a fixed format with a predefined resolution. In case multiple resolutions are to be supported, a duplicate lookup table for each supported resolution should be employed. Also, this restriction only applies to the number of pixels in the image and not the aspect ratio - for example, input images with resolutions 640×480, 480×640 and 1280×240 can be handled using the same lookup table.

Additionally, γ and ε must be fixed - while the former is based on the algorithm itself and will not change at all, the latter is a parameter that, according to the authors of the original work has only a small impact on the detection results [11] and is therefore from now on assumed to use the default value of $\varepsilon = 1$. For the sake of readability, we introduce a new constant t, that incorporates the aforementioned parameters (excluding ε) that are from now on regarded as constants:

$$t := (N \cdot M)^{5/2} \cdot \gamma \tag{2}$$

Moreover, since the lookup table creates entries for every pair of (n,p), the value range for n has to be limited by the introduction of a parameter, which we call n_{max}. The size of the lookup table scales linearly with n_{max}, and the optimization will only be applicable to candidates where $n \leq n_{max}$. For this reason, either n_{max} must be picked very conservatively after extensive experimentation, or a fail-safe must be implemented to ensure the operation of the algorithm if $n > n_{max}$. In our current implementation we switch to the original implementation of the NFA calculation (instead of using the lookup table), if $n > n_{max}$. Usage of heuristics or extrapolation of the lookup table values might be viable strategies as well, although this is a topic of future research.

2) Description: For each pair of (n,p), the Equation 1 can be interpreted as a function of k:

$$\text{NFA}_{n,p}(k) = t \cdot \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j} \tag{3}$$

$\log_{10}(\text{NFA}_{n,p}(k))$ converges towards $\log_{10}(t)$ for $k \to 0$, and as shown in Figure 1, is monotonically falling with rising k.

Our proposed approach is to provide values in a lookup table to approximate $\log_{10}(\text{NFA}_{n,p}(k))$ for every pair of (n,p). This approximation is done piece-wise in three sections, with the breakpoints (being denoted as $k_{low}(n,p)$ and $k_{min}(n,p)$) being part of the lookup table. $k_{min}(n,p)$ is the breakpoint between the second and third section, and is defined to be the lowest k that would yield a successful verification (i.e., Equations 4a and 4b hold):

$$\log_{10}(\text{NFA}_{n,p}(k_{min}(n,p))) \leq \log_{10}(\varepsilon) \tag{4a}$$

$$\log_{10}(\text{NFA}_{n,p}(k_{min}(n,p)-1)) > \log_{10}(\varepsilon) \tag{4b}$$

For any k greater than or equal to $k_{min}(n,p)$, $\log_{10}(\text{NFA}_{n,p}(k))$ is approximated to an arbitrary value smaller than or equal to $\log_{10}(\varepsilon)$. This is justified by the fact that the LSD algorithm does not require the exact NFA value in case the verification of the line-segment is successful.

$k_{low}(n,p)$ is the breakpoint between the first and second section. For any k below $k_{low}(n,p)$, $\log_{10}(\text{NFA}_{n,p}(k))$ is approximated by $\log_{10}(t)$. For any k greater than or equal to $k_{low}(n,p)$, given that it is smaller than $k_{min}(n,p)$, $\log_{10}(\text{NFA}_{n,p}(k))$ is approximated by a quadratic polynomial function with coefficients $c_1(n,p)$, $c_2(n,p)$ and $c_3(n,p)$. A quadratic polynomial was selected since it fits function's form in this section well, allowing for an accurate approximation if the right coefficients are chosen. $k_{low}(n,p)$ is picked so that the total squared approximation error in the first and second section is minimzed, assuming the best polynomial is used for the approximation in each case. If k_{low} is selected too small, the selected polynomial will have to account for the slowly converging area for small k-values, leading to a poor approximation in this area. A too high k_{low} will lead to a good polynomial in the range $k \geq k_{low}$, but high error in the range of k-values slightly below k_{low}. An example demonstration of the relation between k_{low} and the approximation error is shown in Figure 2. The approximation scheme for the different sections is summarized in Table I.

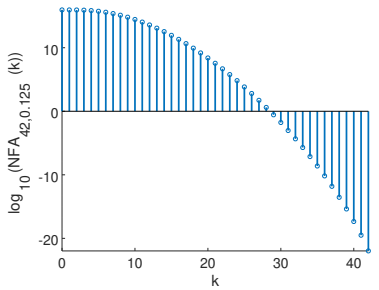


Fig. 1. Plot of $\log_{10}(\text{NFA}_{n,p}(k))$, for $n = 42$, $p = 0.125$ and $N = M = 410$

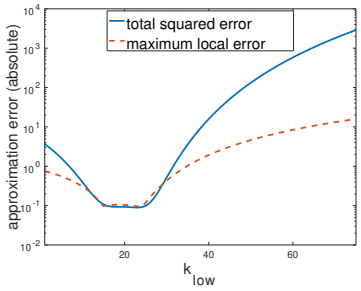


Fig. 2. Total squared approximation error and maximum local error depending on k_{low}, assuming $n = 216$, $p = 0.125$, $N = M = 960$.

TABLE I
APPROXIMATION SCHEME FOR $\log_{10}(\text{NFA}_{n,p}(k))$

Approximation	Section
$\log_{10}(t)$	$0 \leq k < k_{low}(n,p)$
$c_1(n,p)k^2 + c_2(n,p)k + c_3(n,p)$	$k_{low}(n,p) \leq k < k_{min}(n,p)$
$\log_{10}(\varepsilon)$	$k_{min}(n,p) \leq k \leq n$

3) Lookup table generation: Each lookup table entry is a 5-tuple of $k_{min}(n,p)$, $k_{low}(n,p)$, $c_1(n,p)$, $c_2(n,p)$ and $c_3(n,p)$. To generate them, the following steps are performed for every pair of (n,p):

1) Calculate $\log_{10}(\text{NFA}_{n,p}(k))$ starting from $k = 1$ until the result is smaller than or equal to $\log_{10}(\varepsilon)$. The index of the last result is $k_{min}(n,p)$, which is to be stored in the lookup table. All results except for the last one should be stored in an array. If no such $k_{min}(n,p) \leq n$ exists, assume $k_{min}(n,p) := n + 1$.
2) For all $k_{low}(n,p) \in [1, k_{min}(n,p)]$, perform steps 3 and 4.
3) Find a quadratic polynomial to approximate $\log_{10}(\text{NFA}_{n,p}(k))$ in the range $k \in [k_{low}(n,p), k_{min}(n,p) - 1]$, using polynomial regression on the values in the array. Skip this step if $k_{low}(n,p) = k_{min}(n,p)$.
4) Calculate the sum of the squared error values of the approximation for all $k \in [1, k_{min}(n,p)]$. If the approximation quality improved (the total squared error decreased), update $c_1(n,p)$, $c_2(n,p)$, $c_3(n,p)$ and $k_{low}(n,p)$ in the lookup-table.

Algorithm 1: Optimized NFA calculation

inputs : n, k, p
constants: $\log_{10}(\varepsilon)$, $\log_{10}(t)$
output : The approximated NFA value

1 **if** $n = 0$ **or** $k = 0$ **then**
2 **return** $\log_{10}(t)$;
3 $(k_{min}, k_{low}, c_1, c_2, c_3) \leftarrow$ `get_from_LUT`(n,p)
4 **if** $k \geq k_{min}$ **then**
5 **return** $\log_{10}(\varepsilon)$;
6 **if** $k < k_{low}$ **then**
7 **return** $\log_{10}(t)$;
8 $ret \leftarrow c_1 \cdot k^2 + c_2 \cdot k + c_3$;
9 **if** $ret \leq \log_{10}(\varepsilon)$ **then**
10 **return** $\log_{10}(\varepsilon) + \Delta$;
11 **return** ret

TABLE II
COMPARISON OF AVERAGE PROCESSING TIME BETWEEN THE ORIGINAL AND THE PROPOSED ALGORITHM.

	Original	Proposed with $n_{max} =$		
		5000	10000	20000
Processing time	422 ms	365 ms	368 ms	359 ms
Largest improv.	-	27.54%	27.14%	28.8%
Smallest improv.	-	3.48%	1.93%	4.95%
Mean improv.	-	11.46%	10.91%	13.08%

4) Proposed algorithm: Algorithm 1 calculates a single NFA value, given the variables n, k, p, and using the LUT entries $k_{min}(n,p)$, $k_{low}(n,p)$, $c_1(n,p)$, $c_2(n,p)$ and $c_3(n,p)$. The function `get_from_LUT()` fetches an 5-tuple entry from the lookup table. Its implementation depends on the layout of the lookup table, and how it is indexed. Lines 9 and 10 are in place to prevent a false-positive verification, which might happen if the NFA value is approximated by the polynomial to a value below $\log_{10}(\varepsilon)$, usually for a k slightly below k_{min}. Instead, a value of $\log_{10}(\varepsilon) + \Delta$ will be returned, where Δ is the smallest positive value so that $\log_{10}(\varepsilon) \neq \log_{10}(\varepsilon) + \Delta$ using the given floating-point arithmetic.

III. PERFORMANCE RESULTS

1) Processing Time: To measure the performance and the effect of the proposed method, over a wide array of images, the computation time of both the original LSD and the proposed algorithm has been measured on the *TESTIMAGES SAMPLING* set [14] [15], consisting of 40 photographed images. For our experiments, we used a grayscale version of the data set with 1200×1200 resolution and 8 bit intensity depth, with subsampling method labeled `T01R01`. A sample result of the original and proposed algorithm is shown in Figure 3.

The set of images was processed a total of 101 times, with the first processing run of each test image being omitted from the results to account for initial cache misses, branch mispredictions and other transient effects. The mean of the processing time for every image was then calculated. The measurement of the proposed algorithm was repeated for three different values of n_{max} ($n_{max} \in \{5000, 10000, 20000\}$), to evaluate the effect this parameter has on the performance.

The quantitative results are shown in Table II. For all test cases, the proposed algorithm performed better than the original one. The proposed algorithm performed best using $n_{max} = 20k$, with a mean performance increase of 13.08% and offering the largest increase in 39 of the 40 test images (only Test Image #3 performed better using $n_{max} = 5000$). However, the performance difference for the testedvalues of n_{max} is relatively small, with the majority of test cases showing the performance being within $\pm 3\%$ of each other.

2) Detection Quality: Due to the approximation error of the optimization, the rectangle refinement routine of the LSD algorithm potentially selects different precision or width values when advancing through the optimization steps, which can cause detected line-segments to be changed or omitted, as well as new line-segments to be detected.

To measure the impact of the optimization on the detection results, our approach is to try to match each line-segment

detected by the proposed algorithm with an equivalent line-segment detected by the original algorithm, and categorize them using the following metrics:

- **Identical:** Line-segments are categorized as *identical* if they appear in the results of both the original and proposed versions of the LSD algorithm, with identical[1] endpoints.
- **Changed:** Line-segments are categorized as *changed* if a line-segment appears in the results of both the original and proposed versions of LSD algorithms, but the endpoints differ slightly in the results. We qualify a pair of line-segments as *changed* if the sum of distances between their endpoints[1] is smaller or equal to 4 pixels.
- **New:** All line-segments in the results of the proposed algorithm that do not have an *identical* or *changed* match are categorized as *new*.
- **Lost:** All line-segments in the results of the original algorithm that do not have an *identical* or *changed* match are categorized as *lost*.

This categorization was performed on the detection results of both the original and proposed algorithm, using our test image set as the input. For the proposed algorithm, $n_{max} = 20000$ was used, since this version is considered the worst case scenario because it maximizes the usage of the approximated NFA value, and is therefore expected to deviate the most from the original algorithm with regards to the detection results.

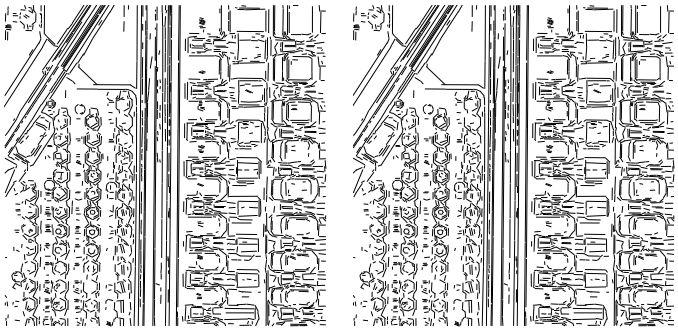
On our test image set, the number of changed line-segments was 0.11% on average, and 0.66% at most. On average, the results of the proposed algorithm contained 0.05% new line-segments, and only four test images suffered from lost line-segments, resulting in less than 0.01% lost line-segments on average across all images, and 0.13% at most. The proposed algorithm yielded only identical results for 19 of the 40 test images. Altogether, on average, 99.84% of the line-segments detected by our algorithm were categorized as *identical*.

The reason for which *new* line segments are present in the proposed algorithm while missing from the output of the original algorithm is that different steps are performed in the rectangle refinement heuristic. Due to approximation error, different sets of rectangle width and precision are used when progressing through the steps of rectangle refinement, which in turn would allow for a successful verification. The approximated verification is designed to never verify a line segment that would not be verifiable using the original calculation. Therefore, since the number of *new* line-segments in our test-image set is higher than the number of *lost* line-segments, we contend that the approximation proposed here actually improves the detection quality of the original LSD algorithm.

IV. Hardware Implementation

While hardware implementations of line-segment detection algorithms have been proposed [16] [17], their detection mechanisms differ from LSD - for example, in [16] detection of line-segments within curved structures is drastically different compared to LSD. For this reason, software applications using LSD for line-segment detection cannot easily switch to a different algorithm for hardware implementation, without incurring considerable engineering effort and testing. Therefore, we are working on hardware accelerated line-segment detection algorithms which are based on LSD and offer a high degree of parity on the detection results.

[1]The coordinates of the line-segment endpoints were rounded to the nearest integer beforehand.



(a) (b)

Fig. 3. Detection result of LSD (a) without and (b) with LUT-based approximation (Test image 38).

One of the major issues that make hardware acceleration of LSD difficult is the usage of mathematical operations are considered to be costly to implement in hardware, both regarding area usage as well as latency they introduce. While sine and cosine functions are used in the earlier stages of the algorithm (e.g. the creation of the regions), the math used to calculate the NFA value is the most complex, employing the hyperbolic sine, logarithm and exponential functions, with the latter two of them being used in a loop that is executed up to n times per NFA calculation. In our proposed algorithm, these operations are replaced by fetching values from a lookup table as well as a series of comparisons, additions and multiplications, resulting in faster operation and less combinatorial logic required. However, the proposed algorithm does introduce a strict requirement on memory to store the lookup table. This can be either on-chip, off-chip or both. For instance, if the on-chip memory is not large enough to contain the entire table, storing values for low n values that are more commonly used on-chip, and storing values for high n values that are only rarely accessed off-chip, would be sufficient.

V. Conclusion

In this paper, we proposed an improved version of the LSD algorithm. Te proposed algorithm approximates the calculation of the NFA value by a series of lookup table searches, improving the processing time by 13.08% on average. We have shown that over 99.8% of the detected line-segments are identical in both the proposed and original LSD algorithm, and that the detection quality slightly increased in the proposed algorithm. Finally, we have discussed implications of the proposed algorithm on reducing the efforts to develop a hardware implementation of LSD.

REFERENCES

[1] K. Li *et al.* Joint point and line segment matching on wide-baseline stereo images. In *2016 IEEE WACV*, pp. 1–9, 2016.

[2] S. Lin *et al.* Monocular vision-based real-time target recognition and tracking for autonomously landing an uav in a cluttered shipboard environment. *Autonomous Robots*, 41, 04 2016.

[3] Y. Zhou *et al.* Svo-pl: Stereo visual odometry with fusion of points and line segments. In *2018 IEEE ICMA*, pp. 900–905, 2018.

[4] C. Creusot and A. Munawar. Low-computation egocentric barcode detector for the blind. In *2016 IEEE ICIP*, pp. 2856–2860, 2016.

[5] T. Santos *et al.* Plined: Vision-based power lines detection for unmanned aerial vehicles. In *2017 IEEE ICARSC*, pp. 253–259, 2017.

[6] J. Duan *et al.* Lane line recognition algorithm based on threshold segmentation and continuity of lane line. In *2016 2nd IEEE ICCC*, pp. 680–684, 2016.

[7] S. Liu *et al.* Effective road lane detection and tracking method using line segment detector. In *2018 37th CCC*, pp. 5222–5227, 2018.

[8] Tao Ma et al. A sea-sky line detection method based on line segment detector and hough transform. In *2nd ICCC*, pp. 700–703, 2016.

[9] U. BUDAK *et al.* Automatic airport detection with line segment detector and histogram of oriented gradients from satellite images. In *2018 International Conference on Artificial IDAP*, pp. 1–5, 2018.

[10] N. Liu *et al.* Airport detection in large-scale sar images via line segment grouping and saliency analysis. *IEEE Geoscience and Remote Sensing Letters*, 15(3):434–438, 2018.

[11] R. Grompone von Gioi *et al.* LSD: a Line Segment Detector. *Image Processing On Line*, 2:35–55, 2012.

[12] A. Desolneux *et al.* Meaningful alignments. *International Journal of Computer Vision*, 40(1):7–23, Oct 2000.

[13] A. Delsolneux *et al.* *From Gestalt Theory to Image Analysis: A Probabilistic Approach*, volume 34. Springer, 01 2008.

[14] N. Asuni and A. Giachetti. TESTIMAGES: A large data archive for display and algorithm testing. *Journal of Graphics Tools*, 17:113–125, 10 2013.

[15] N. Asuni. TESTIMAGES/SAMPLING. URL: https://testimages.org/sampling (Accessed: 10-23-2020).

[16] F. Zhou *et al.* Fast and resource-efficient hardware implementation of modified line segment detector. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(11):3262–3273, Nov 2018.

[17] A. Abdallah *et al.* Fpga implementation of resistor network for fast segment line detector. In *2017 29th ICM*, pp. 1–4, 2017.