Pacific Graphics 2019
C. Theobalt, J. Lee, and G. Wetzstein
(Guest Editors)

Volume 38 (2019), Number 7

Inertia-based Fast Vectorization of Line Drawings

Patryk Najgebauer [1] and Rafał Scherer[1]

Częstochowa University of Technology
Al. Armii Krajowej 36, 42-200 Częstochowa, Poland
e-mail: patryk.najgebauer@iisi.pcz.pl, rafal.scherer@iisi.pcz.pl

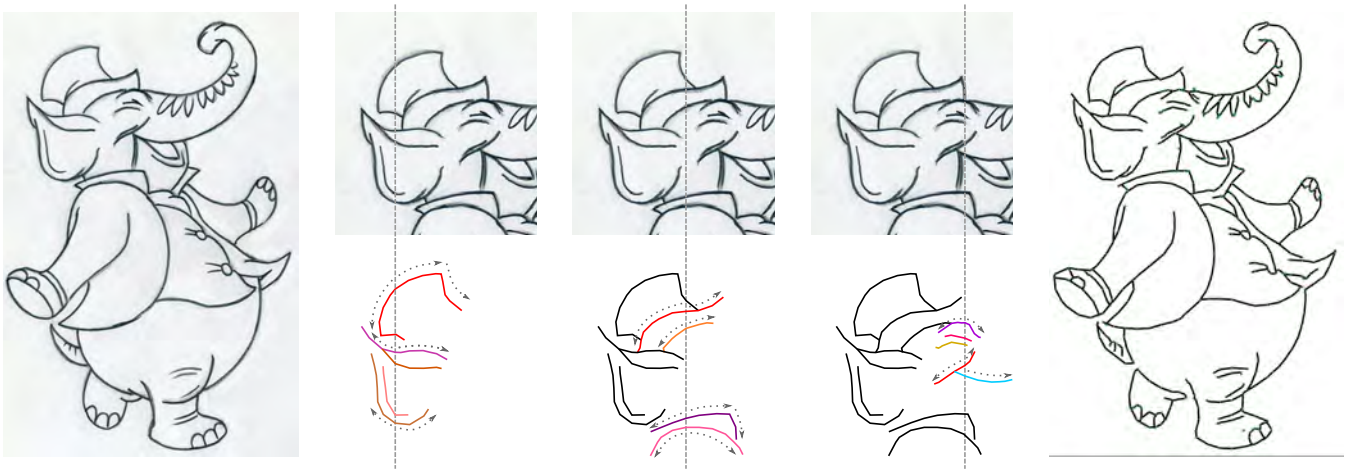

Figure 1: *Sketch vectorization starts from detecting intersections between sketch and grid lines. The algorithm use them as the starting points to trace lines with the moment of inertia to resolve best line continuation at any line junction. With each next grid line (three examples in the figure), the method detects more lines that finally map the whole sketch to lists of lines where each line is a list of nodes. This approach produces finally vectorized lines in a single pass during sketch tracing without additional subprocesses. The algorithm is described in details in Section 2.*

Abstract

Image vectorisation is a fundamental method in graphic design and is one of the tools allowing to transfer artist work into computer graphics. The existing methods are based mainly on segmentation, or they analyse every image pixel; thus, they are relatively slow. We introduce a novel method for fast line drawing image vectorisation, based on a multi-scale second derivative detector accelerated by the summed-area table and an auxiliary grid. Image is scanned initially along the grid lines, and nodes are added to improve accuracy. Applying inertia in the line tracing allows for better junction mapping in a single pass. Our method is dedicated to grey-scale sketches and line drawings. It works efficiently regardless of the thickness of the line or its shading. Experiments show it is more than two orders of magnitude faster than the existing methods, not sacrificing accuracy.

CCS Concepts

• ***Computing methodologies*** → *Non-photorealistic rendering; Parametric curve and surface models;*

1. Introduction

Image vectorizations methods are one of the oldest computer vision algorithms, embedded in most vector image processing software. Their goal is to transform a raster (bitmap) image into a set of vectors to allow easier editing, scalability and, usually, compactness. Automatic vectorization tools save much time of manual redrawing during a workflow. Vectorisation can have different aims. In some cases, we need to extract gradient-filled shapes from an image; in other cases, only shape outlines, or a sketch line. Despite their wide adoption, industrial vectorization is performed manually, and it takes less time than manual corrections after automatic

processing. Therefore, there is ongoing work on making vectorization algorithms more human-like intelligent. Even for clean, noise-free line drawings, it is still challenging to disambiguate line junctions. The significant advantage of vectorization compared to simple scanning is the resultant image quality that allows to resize or manipulate image without the quality loss, unlike in the case of raster images. The second advantage is memory usage as the vectorized image needs significantly fewer data than a similar bitmap. For real image-like photography, or painting vectorization it is not an optimal solution because of the image complexity but in most cases of drawing or sketch it is a perfect tool to import line drawing images or sketches for further processing.

Image vectorization can be divided into two main classes by the purpose. The first one is a family of industrial applications that require accuracy of the final vectorized image, and the second one is for art purposes that is more focused on visual effect than the accuracy of the image mapping.

In the first case, most of the work is performed manually, especially for technical drawings that need a precise description of the subject to which they refer. This type of drawing requires some expert knowledge; thus, simple vectorization of technical drafts often does not bring the desired effect.

In the second case, the requirements are different. Art drafts often describe more impressions than reality. Most art drawings depict non-photographic content that is easily interpreted by human perception. In most cases, vectorization is used to import drafts or fragments of real images from photos in a digital graphics workflow. Vectorized content is easy to manipulate, edit and combine with other content into the final product. Vector graphics compared to bitmaps makes an impression of pure, clean design and is scalable which is often used in an advertisement, animation, or web development starting from brand representation, through graphs to the entire layout.

Most of vector graphic software has own implementation of vectorization tool, e.g. Adobe Illustrator Image Trace [Tea12] or Inkscape Potrace [Sel03]. These algorithms are based on a single channel image segmentation. In the literature, there were introduced more advanced algorithms based on segmentation and boundary fitting [HDS*18]; some algorithms start to vectorize gradient shape fill [SLWS07; OBW*08; LHFY12]. To provide better results or realistic output images, some works proposed image decomposition into overlapped transparent layers [FLB17; TDSG15; TLG17; AASP17]. In [BS18] images are vectorized by frame field processing. The algorithm from [NHS*13] analyzes the drawing's topology in order to resolve junction ambiguities. In [FLB16], the authors optimize a balance between fidelity and simplicity, yet, the algorithm is much slower than the algorithm presented in this paper. In [XLY09] an algorithm for image vectorization is proposed aimed at high quality photograph representation. In [KL11] smooth vector representation is created from very low-resolution bitmap images (icons, sprites etc.). Cartoon images are vectorized by image triangulation in [ZY01] and in [FSH11] by high order thin-plate splines. Both methods are aimed at accuracy not speed.

In the paper, we focus on the efficient handmade sketch and line drawing vectorization. The proposed method is more than two orders of magnitude faster than the existing methods with similar accuracy. The speed comes from applying the auxiliary grid and the summed-area table [Cro84]. At the same time, the proposed vectorization acts similarly to human perception with the use of inertia. The main contribution of the paper is as follows:

- We present a novel method for sketch vectorization that is based on the auxiliary grid. Image scanning to determine intersection points between the sketch and the auxiliary grid lines allows to speed up the sketch vectorization process. As a result, the method omits most of the image and focuses only on the sketch line.
- Multiscale second derivative operators in combination with the summed-area table, bring a significant acceleration of the method, and it is irrelevant to the width of the sketch line.
- Adding inertia to the sketch line tracing allows for better junction handling in a single pass during line trace.

The paper is organized as follows. Section 2.1 describes the proposed method of the multiscale line detection based on the second derivative. Section 2.2 introduces the auxiliary grid used to detect intersection points with the sketch lines. Section 2.3 shows a method of sketch line tracing to determine other intersection points starting from the first detected intersection with the auxiliary grid. Node combining and insertion of additional nodes on a detected bend are described in Section 2.4. Section 3 provides simulation results on handmade sketches and comparison to similar existing algorithms.

2. Proposed Method

The presented method is based on line detection by the second derivative operator instead of image segmentation proposed in the literature. On input, the method takes grayscale sketch image and computes the summed-area table [Cro84] to speed up line detection. The summed-area table computes fast a sum of pixels in a given area. Our algorithm in a single pass traces lines and creates nodes with some inertia that mimics the sketcher's movements. This approach allows to speed up the vectorization process and reduce the problem of wrong node connection at junctions. Misinterpreted junctions make a prominent wrong impression concerning the vectorized sketch. To deliver a better node distribution, our method places most of the nodes at the intersections between sketch and additional added lines of the auxiliary grid. The auxiliary grid covers the entire sketch, and nodes placed on its lines create a better visual effect, especially in the case of close parallel curves. We present the algorithm pipeline in Figure 2, and an example of the algorithm run for three consecutive grid lines in Figure 1.

2.1. Multiscale Line Detection

The standard filter masks used in general applications have a disadvantage in the form of its fixed size. Therefore, we use the second derivative filter as a line detector (Figure 4). Small operators do not overlap the line and detect wide lines as two separate adjacent lines. In order to avoid this problem, we apply scalable operators that are adjusted to the detected line. After finding a line, the method gradually increases coefficient λ, which defines the filter size and compares the response to obtain the best-fitted filters that overlap the line (Figure 3). For a bitmap consisting of thin lines, λ has low

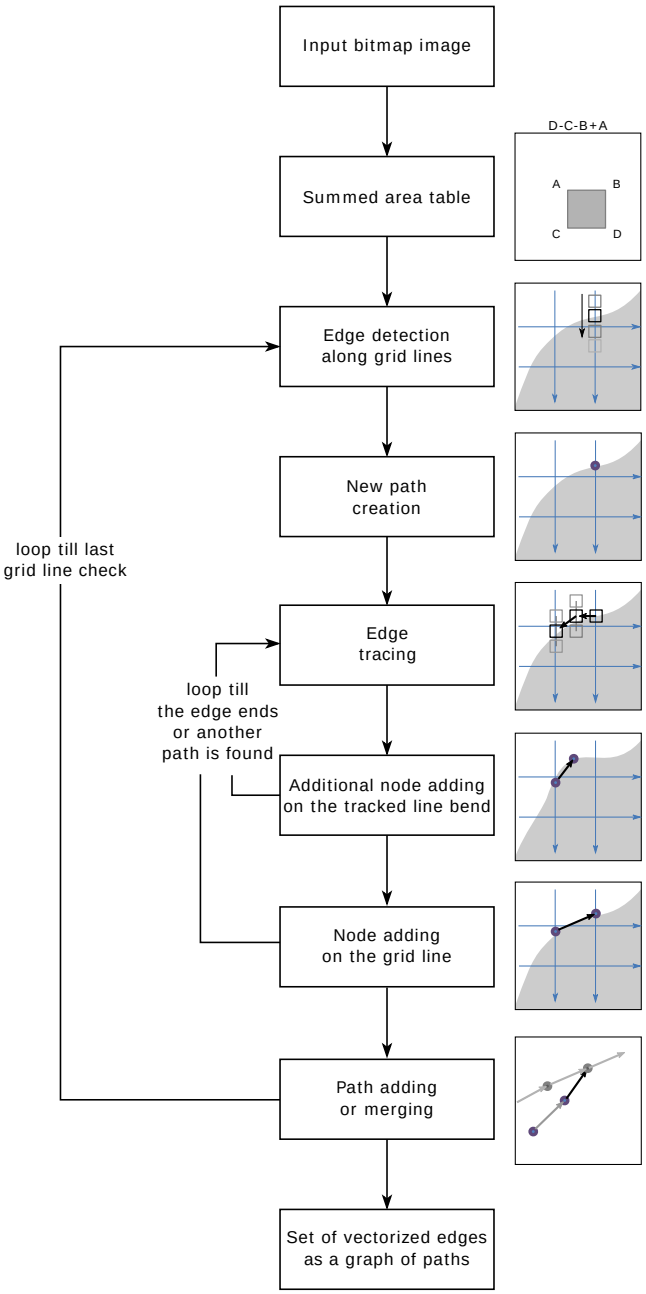

Figure 2: *The general pipeline of the algorithm presented in the paper.*

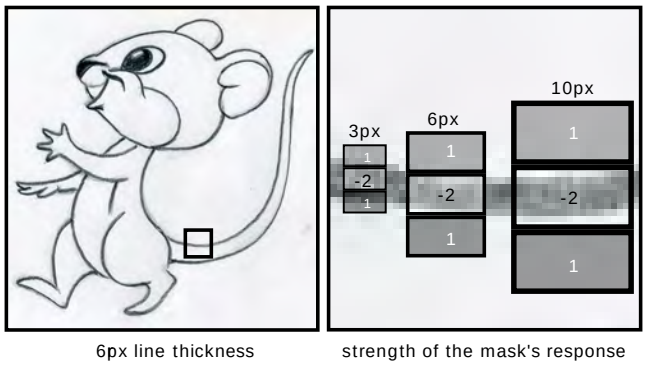

Figure 3: *The multi-scale second derivative filter used by the method. During intersection detection, the algorithm starts convolution with the smallest mask, and after the positive response, the filter size is gradually increased to overlap the sketch line, i.e. to obtain the strongest mask response (3 px < 6 px > 10 px).*

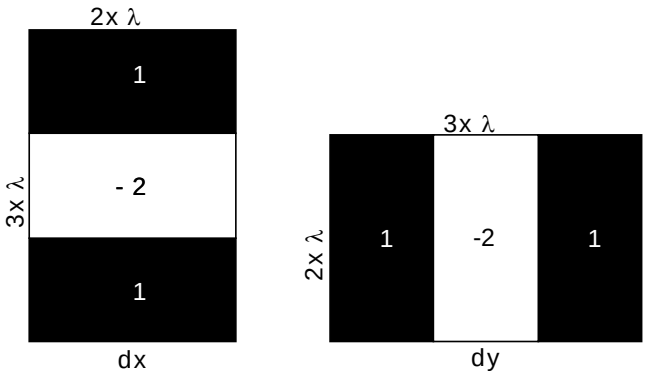

Figure 4: *Scalable second derivative filters used in the proposed method. The filter size coefficient λ is adjusted to the line thickness. The properly adjusted filter that overlaps the line width yields the highest convolution response.*

value, and thick or jagged lines need higher values of λ. This is illustrated in Figure 14, where some adjacent lines are vectorised as as a single line.

The range of filter size coefficient λ that the method uses is adjustable depending of the type of the sketch (i.e. image resolution, line width or shading). Setting a higher λ upper limit allows detecting filled sketch primitives, e.g. eyes or a nose. Figure 5 presents an example of this feature. The mask response is weaker because half of the mask correlates with the boundary of the primitive. The high contrast of the sketch causes that the mask response is sufficient to detect the boundary line. As the operator response, in each point of the image, we obtain two-dimensional vector $\vec{v} = [x, y]$ where component x and y correspond to filter dx and dy (Fig. 4). The filter response is normalized in the range from -1 for dark areas and to 1 for light areas regardless of the mask size. In the case of the point over a solid and unchanging background, the result of the operator will be equal to 0. In each image that represents the operator responses (feature map), we show only mask responses where the component sum of $\vec{v}$ is less than zero.

2.2. Auxiliary grid

In order to speed up line detection and to obtain better results of node positioning, we introduced to our method the auxiliary grid. The auxiliary grid is used as a template to determine the intersec-

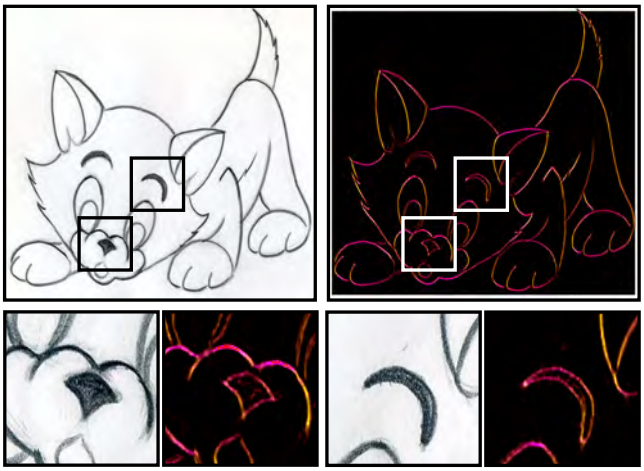

Figure 5: *An advantage of the second derivative mask is that it allows detecting the boundary line of the filled primitive. Yellow color represents component x that corresponds to dx mask and pink color y component that corresponds to dy.*

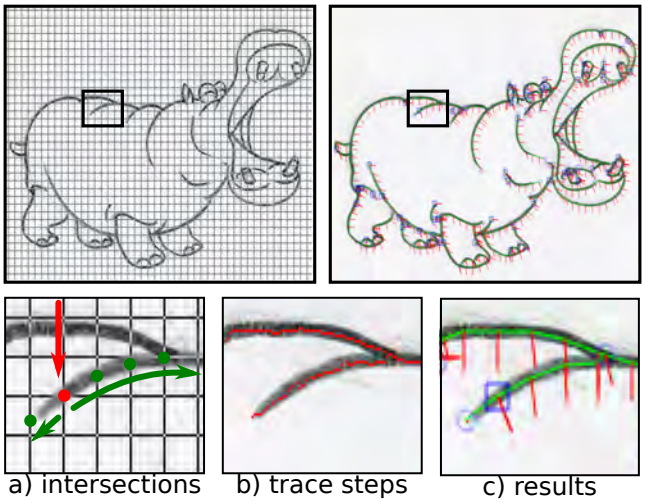

Figure 6: *Line localization and tracing with the auxiliary grid using. a) Line tracing start point localization (the first step) along the grid line (red arrow $\vec{d}_g$ direction and red dot as started nodes) and (second step) line tracing (green arrow directions and green dots as next nodes of the traced line). b) The real route traversed by the algorithm during the line trace (red pixels) that follows pixel by pixel along local minima. c) Results: blue rectangle – starting node, blue circle – ending node, red lines – normal vector $\vec{n}$ for node, green lines – node connections.*

tion points between the sketch and the grid lines. Our method initially provides line detection for points under grid lines and bypasses the rest of the image. Figure 6 presents an example of a 15-pixel grid. That means that the method checks only one pixel per 15 pixels of the image to localize the sketch lines. The distance between grid lines is adjustable, and in the case of making it sparser, the method speeds up the vectorization process, but it might miss some short lines and details of the image (Fig. 12). The method checks each horizontal and vertical grid line across the image and detects local minima (Fig.6a, red line). When the method detects an intersection and checks that the current point is not already traced then it adjusts the mask size and starts trace line in both directions (green line). During line trace, the method gradually adds additional nodes on each intersection with the grid lines (green points).

During node insertion, the method is limited by a single condition. Nodes are created only if the normal vector $\vec{n} = |\vec{v}|$ for the sketch line at the intersection point is in the proper direction $\vec{d}_g$ to the grid line that fulfils the condition $|\vec{n} \cdot \vec{d}_g| < \cos 30^\circ$. This condition prevents from inserting multiple nodes in the case where the sketch line coincides with the grid line instead of intersecting it. Nodes inserted at grid lines provide a better node distribution between the lines. Such vectorization looks cleaner on adjacent lines in contrast to inserting nodes over the lines at a certain fixed distance.

2.3. Edge tracking

After detecting a sketch line and setting the starting node at position $\vec{p}$, the method follows the line to insert new nodes in two directions. The starting direction $\vec{d}$ is horizontal or vertical and depends on $\vec{n}$ of the initial node. For a horizontal line, vector $\vec{d}$ becomes $[-1,0]$ for the first tracing direction and $[1,0]$ for the second. For a vertical line, respectively $[0,-1]$, $[0,1]$.

In each step, the method moves the position of $\vec{p}$ by a single pixel in the direction of vector $\vec{d}_{l5}$ that is equal to the average value of the last five values of $\vec{d}$. At the beginning, vector $\vec{d}_{l5}$ equals vector $\vec{d}$. We checked empirically that a lower value than 5 causes that the inertia of tracing is too small. Higher value could cause tracing to leave the current line.

After moving the position of $\vec{p}$, the method corrects its position by determining the local minimum in the vertical direction to the last move. The method checks neighbour point at the two-pixel distance in each direction and changes $\vec{p}$ if the operator response in the checked point is stronger than of the current point. Figure 7 shows the trace process when, after each step, the method corrects the current position. To resolve the problem of overlapping already detected lines, we use an additional map from the previous steps.

Following position correction, the method determines the direction of the last step $\vec{d} = \vec{p_{cur}} - \vec{p_{last}}$ and updates $\vec{d}_{l5} = (\vec{d}_{l5} * 4 + \vec{d})/5$. Vector $\vec{d}_{l5}$ value reacts on the line direction change weaker but is corrected after each step. This allows to trace lines more smoothly and on the sketch line junctions, the trace follows through the junction to look for a straight continuation of the line. Figures 6b and 7 show the mapped path of tracings with red pixels. As we can see, the tracing does not change its direction rapidly during abrupt line width change. In other words, the method tries to follow the pencil path. Method traces the line until:

- it encounters another already traced line found in the additional trace map,

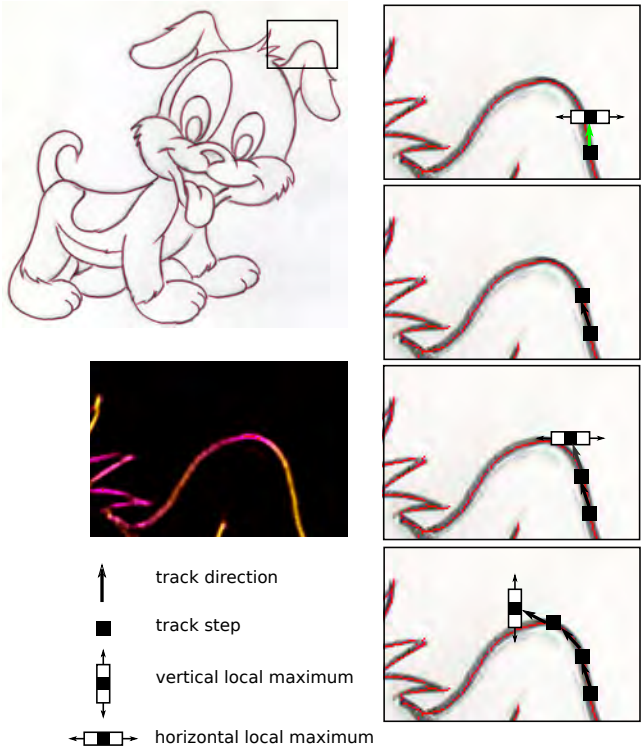

Figure 7: *Line tracing correction. After each step, the algorithm corrects the position; thus it follows exactly the middle of the sketch line.*

- the sum of $\vec{v}$ components for the current traced point falls under the threshold (that means the line is too weak).

When the trace interrupts the current line, the method inserts the ending node and returns to search for the next intersection point with the grid line to start another tracing. For very subtle non-contrasting sketch lines, the threshold value should be smaller. However, this will make the algorithm more sensitive to small noise elements.

2.4. Node combining

We could describe the result of the line vectorization process as a graph of nodes connected by edges. However, in each traced path that represents a single line, each node is always connected only with two other nodes. Thus, to store the results more efficiently compared to a traditional graph, we combine a set of single line nodes into a single list of nodes. Nodes are ordered in a list according to their detection during tracing; thus, we do not require the information about graph edges. As a result, the method operates on lists of lines where each line is a list of nodes. This approach produces finally vectorized lines in a single pass during sketch tracing without additional subprocesses.

2.5. Additional nodes – line bend detection

During line tracing the nodes inserted on the detected intersection between the sketch and grid lines could not correctly reproduce the details of the image. In order to resolve this problem, we have to add some additional nodes. To locate these points for additional nodes, the method examines bends of the traced line. Bend detection relies on comparing two vectors $\vec{d}1$ and $\vec{d}2$ that are determined in the same way as $\vec{d}_{l5}$. Both vectors are equal to the average value of $\vec{d}$ from the last steps. The number of steps is adjustable for both vectors. After each step, we calculate the vectors according to formula

$$\vec{d}_i = (\vec{d}_i * steps + \vec{d})/(step + 1)\,, \tag{1}$$

where i is the vector number and $step$ is the number of the last step counted. In practice, $step$ determines the strength with the current $\vec{d}$ affects $\vec{d}_i$.

Finally, each vector is normalized and their dot product is compared

$$|\vec{d}_1 \cdot \vec{d}_2| < \cos 15^\circ\,. \tag{2}$$

If the dot product is less then threshold equal to $\cos 15^\circ$ then the method adds an additional node and resets the vectors $\vec{d}_1 = \vec{d}_2 = \vec{d}$. Figure 8 presents a graphic explanation of the bend detection. The value of $\vec{d}_1$ changes faster than $\vec{d}_2$ during tracing. When the traced line starts to change direction, then the dot product between $\vec{d}_1$ and $\vec{d}_2$ start decreasing and at last, exceeds the threshold. Moreover, Figure 8, in the third column, presents an example of invalid bend detection. In this case, if $step$ equals 1 for $\vec{d}_1$, then $\vec{d}_1$ changes rapidly between steps what is caused by noise. In the case of $step = 2$, the bend points are detected more correctly.

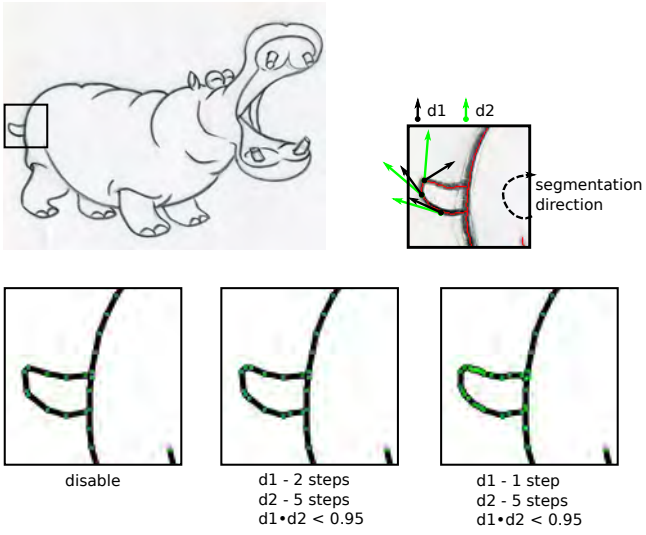

Figure 8: *Line bend detection by comparing trace direction vectors $\vec{d}1$ and $\vec{d}2$. Each vector is equal to the average value of $\vec{d}$ vectors from several last trace steps. The first column shows the results without bend detection, the second one with $\vec{d}1$ counted from the last two steps and $\vec{d}2$ from the last five steps. The last column is an example for single step and five last steps.*

3. Experiments

Experiments were performed on an image set of handmade sketches without any preprocessing. The method was implemented as a single thread application in C++ using Qt and STL libraries. Experiments were performed on a Linux machine with AMD APU processor with 16GB memory. The proposed method was run with parameters: grid size = 15px, mask size range = [2px,5px], treshold = 0.13, steps $d_1 = 2$ and $d_2 = 5$.

Table 1 presents the speed results of the method compared to other algorithms. The aim was to compare the proposed method with similar methods from the literature. For this purpose, due to the lack of access to the author's implementation of the referenced algorithms, we presented a summary of the results of our method with the results presented in the compared articles. In the compared papers, a common large dataset was not defined, and the focus was on the assessment of the visual effects obtained by the methods. The tests were carried out on a small collection of hand-made sketches generally available on the page http://www-labs.iro.umontreal.ca/~bmpix/ [BS18]. As we can see, our method thanks to its single pass vectorization and the auxiliary grid line detection significantly accelerates the vectorization process. According to Table 2, our method performs a large part of the calculation only for the traced image lines determined as the local minimum. The compared methods operate on the entire set of dark pixels.

When we compare the number of lines in Table 2 with the experimental results in Figure 9 and Fig. 10 we can have an impression that there are too many of them. It is caused by the details of images where the method creates many short lines to map elements properly, which can be observed in Figure 6. Compared to other methods in Fig. 9, our method does not create improper connections between lines like it is in the case of Favreau et al. [FLB16] method. Our method does not go beyond the sketch lines during the trace. Moreover, we can resolve the problem of filled primitives. Thanks to the second derivative operator, the presented method detects the outline of the primitives, e.g. in the case of the cat nose in Figure 10.

Some weaknesses of our method compared to Bessmeltsev et al. [BS18] can be seen at the jagged lines like animal hair in the images. This weakness is caused by the grid line that does not intersect with these small elements; however, our method reproduces these details but with the lack of some connections between them. Figure 12 presents more precisely the problem of details missing with the grid size increasing. As we can see, the best results are obtained for 10- and 20-pixel grids. In the case of a 40 pixel grid, we obtain a noticeable loss of parts of the sketch that were not intersected by the grid.

In order to specify the precision of the bitmap-vector mapping, we used the method of determining the coverage ratio of the original sketch by a vectorized representation. This form is not entirely accurate, because, in the case of sketches, we have lines of different thickness, which will not always be fully covered by lines of a vector image with a fixed width. To more accurately determine the precision of the method, we compare the vector image with the image representing the second derivative of the input image. An example is shown in Figure 11. The presented method uses second derivative filters to locate the lines, so the presented comparison best reflects the accuracy of the method, as well as allows for a better determination of accuracy in the case of filled areas, where the method is to determine the outline of this area.

The average precision of vectorization for a set of sketches consisting of 90 images was 81 %. More detailed results are presented in Table 2 for sample sketches from Figure 10. The method loses accuracy for irregularly shaded, or hatched fill pattern images, e.g. penguin or trees in Fig. 13. In these areas, the second derivative operator returns positive results, however they are not lines but shadowing fragments, which then lead to the determination of additional lines during vectorization.

For an additional experiment, for vector images from the CorelDraw clipart collection, the method obtained a higher average precision of 89 %. This is due to the lack of irregularly shaded fields, which add false lines. Sample results are shown in Figure 14. The method omits some of the details because it is oriented towards black lines. Thus, some of the edges, particularly coloured elements, have been omitted. In addition, in the case of thin elements, the method can present them with one line instead of two in parallel, this is due to the size of the mask, which can cover the whole of a given fragment.

4. Conclusion

The proposed method considerable speeds up the vectorization process of sketch and line drawing images compared to the existing solutions. Application of the auxiliary grid to sketch vectorization accelerates line detection which is confirmed by the experimental results. The adjustable value of the grid size can be used in two ways. Increasing the size of the grid reduces the number of details, at the same time accelerating the method. On the other hand, it also can be used as a filter to detect significant lines of the sketch. Very good handling of line intersections is obtained by the application of inertia, similar to an artist or a designer strokes. In further studies, we will aim at overcoming the problem of losing the details caused by increasing the grid size by designing better junction detection. In most sketch and line drawing examples, lines are interconnected or adjacent and such places should be even more precisely detected than in the case of the presented approach. By applying even more intelligent line intersection handling, the method could be sped up significantly by using a much larger grid.

Acknowledgement

The project financed under the program of the Polish Minister of Science and Higher Education under the name "Regional Initiative of Excellence" in the years 2019–2022 project number 020/RID/2018/19, the amount of financing 12,000,000.00 PLN.

References

[AASP17] AKSOY, YAĞIZ, AYDIN, TUNÇ OZAN, SMOLIĆ, ALJOŠA, and POLLEFEYS, MARC. "Unmixing-based soft color segmentation for image manipulation". *ACM Transactions on Graphics (TOG)* 36.2 (2017), 19 2.

[BS18] BESSMELTSEV, MIKHAIL and SOLOMON, JUSTIN. "Vectorization of Line Drawings via PolyVector Fields". *arXiv preprint arXiv:1801.01922* (2018) 2, 6, 8.

Table 1: *Comparison of algorithm statistics and times*

	input resolution	number of dark pixels	Noris et al. time	Favreau et al. time	Bessmeltsev et al. time	our time
Puppy	660 x 624	29908	26 s	224 s	41 s	0.08 s
Hippo	700 x 535	25114	24 s	120 s	43 s	0.06 s
Banana Tree	589 x 865	18619	15 s	244 s	23 s	0.08 s
Penguin	500 x 714	24134	23 s	181 s	56 s	0.06 s
Kitten	700 x 554	29023	38 s	250 s	81 s	0.07 s
Elephant	500 x 753	34569	33 s	270 s	55 s	0.07 s

Table 2: *Additional statistics for the proposed method*

	summed-area table preparing time [s]	vectorization time [s]	segmentation steps	number of lines	number of nodes	sketch coverage precision
Puppy	0.014	0.062	6817	156	1324	86 %
Hippo	0.008	0.053	5054	134	1188	81 %
Banana Tree	0.017	0.067	5945	106	996	92 %
Penguin	0.014	0.048	3858	50	625	71 %
Kitten	0.012	0.057	5780	138	1170	86 %
Elephant	0.009	0.061	7416	194	1516	88 %

[Cro84] CROW, FRANKLIN C. "Summed-area tables for texture mapping". *ACM SIGGRAPH computer graphics* 18.3 (1984), 207–212 2.

[FLB16] FAVREAU, JEAN-DOMINIQUE, LAFARGE, FLORENT, and BOUSSEAU, ADRIEN. "Fidelity vs. simplicity: a global approach to line drawing vectorization". *ACM Transactions on Graphics (TOG)* 35.4 (2016), 120 2, 6, 8.

[FLB17] FAVREAU, JEAN-DOMINIQUE, LAFARGE, FLORENT, and BOUSSEAU, ADRIEN. "Photo2clipart: image abstraction and vectorization using layered linear gradients". *ACM Transactions on Graphics (TOG)* 36.6 (2017), 180 2.

[FSH11] FINCH, MARK, SNYDER, JOHN, and HOPPE, HUGUES. "Freeform Vector Graphics with Controlled Thin-plate Splines". *ACM Trans. Graph.* 30.6 (Dec. 2011), 166:1–166:10. ISSN: 0730-0301. DOI: 10.1145/2070781.2024200. URL: http://doi.acm.org/10.1145/2070781.2024200 2.

[HDS*18] HOSHYARI, SHAYAN, DOMINICI, EDOARDO ALBERTO, SHEFFER, ALLA, et al. "Perception-driven semi-structured boundary vectorization". *ACM Transactions on Graphics (TOG)* 37.4 (2018), 118 2.

[KL11] KOPF, JOHANNES and LISCHINSKI, DANI. "Depixelizing pixel art". *ACM Transactions on graphics (TOG)* 30.4 (2011), 99 2.

[LHFY12] LIAO, ZICHENG, HOPPE, HUGUES, FORSYTH, DAVID, and YU, YIZHOU. "A subdivision-based representation for vector image editing". *IEEE transactions on visualization and computer graphics* 18.11 (2012), 1858–1867 2.

[NHS*13] NORIS, GIOACCHINO, HORNUNG, ALEXANDER, SUMNER, ROBERT W, et al. "Topology-driven vectorization of clean line drawings". *ACM Transactions on Graphics (TOG)* 32.1 (2013), 4 2.

[OBW*08] ORZAN, ALEXANDRINA, BOUSSEAU, ADRIEN, WINNEMÖLLER, HOLGER, et al. "Diffusion curves: a vector representation for smooth-shaded images". *ACM Transactions on Graphics (TOG)*. Vol. 27. 3. ACM. 2008, 92 2.

[Sel03] SELINGER, PETER. "Potrace: a polygon-based tracing algorithm". *In http://potrace.sourceforge.net*. 2003 2.

[SLWS07] SUN, JIAN, LIANG, LIN, WEN, FANG, and SHUM, HEUNG-YEUNG. "Image vectorization using optimized gradient meshes". *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, 11 2.

[TDSG15] TAN, JIANCHAO, DVOROŽŇÁK, MAREK, SYKORA, DANIEL, and GINGOLD, YOTAM. "Decomposing time-lapse paintings into layers". *ACM Transactions on Graphics (TOG)* 34.4 (2015), 61 2.

[Tea12] TEAM, ADOBE CREATIVE. *Adobe Illustrator CS6 Classroom in a Book*. Adobe Press, 2012 2.

[TLG17] TAN, JIANCHAO, LIEN, JYH-MING, and GINGOLD, YOTAM. "Decomposing images into layers via RGB-space geometry". *ACM Transactions on Graphics (TOG)* 36.1 (2017), 7 2.

[XLY09] XIA, TIAN, LIAO, BINBIN, and YU, YIZHOU. "Patch-based Image Vectorization with Automatic Curvilinear Feature Alignment". *ACM Trans. Graph.* 28.5 (Dec. 2009), 115:1–115:10. ISSN: 0730-0301. DOI: 10.1145/1618452.1618461. URL: http://doi.acm.org/10.1145/1618452.1618461 2.

[ZY01] ZOU, JU JIA and YAN, HONG. "Cartoon image vectorization based on shape subdivision". *Proceedings. Computer Graphics International 2001*. IEEE. 2001, 225–231 2.

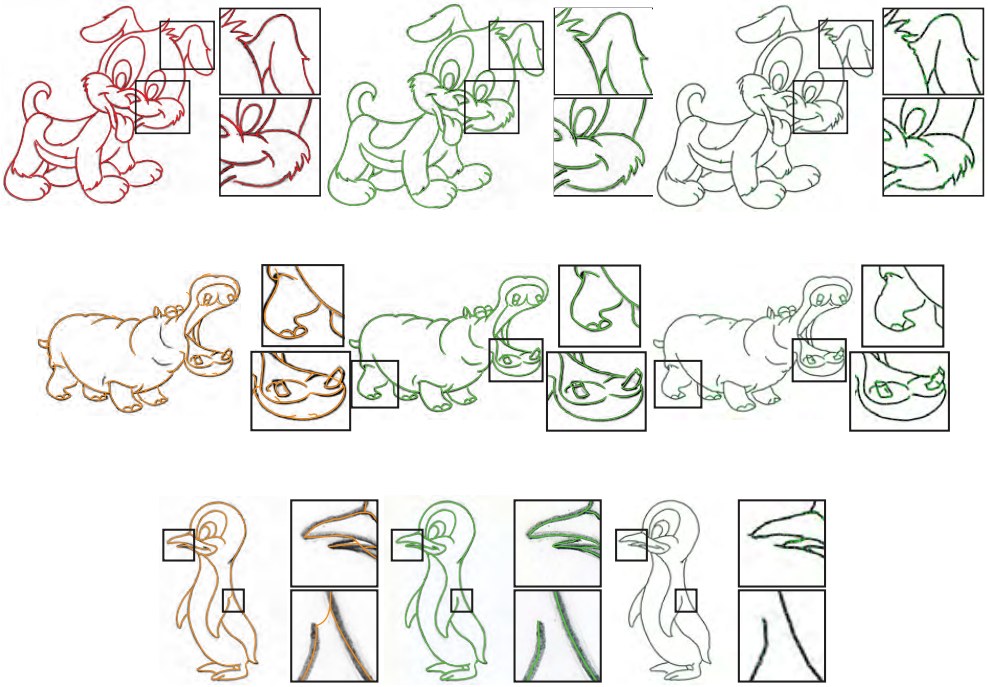

Figure 9: *Experimental results from Table 1. The first column presents the method by Favreau et al. [FLB16], the second Bessmeltsev et al. [BS18] and the third one – our method.*

Figure 10: *Several additional examples of the experimental results from Table 1.*

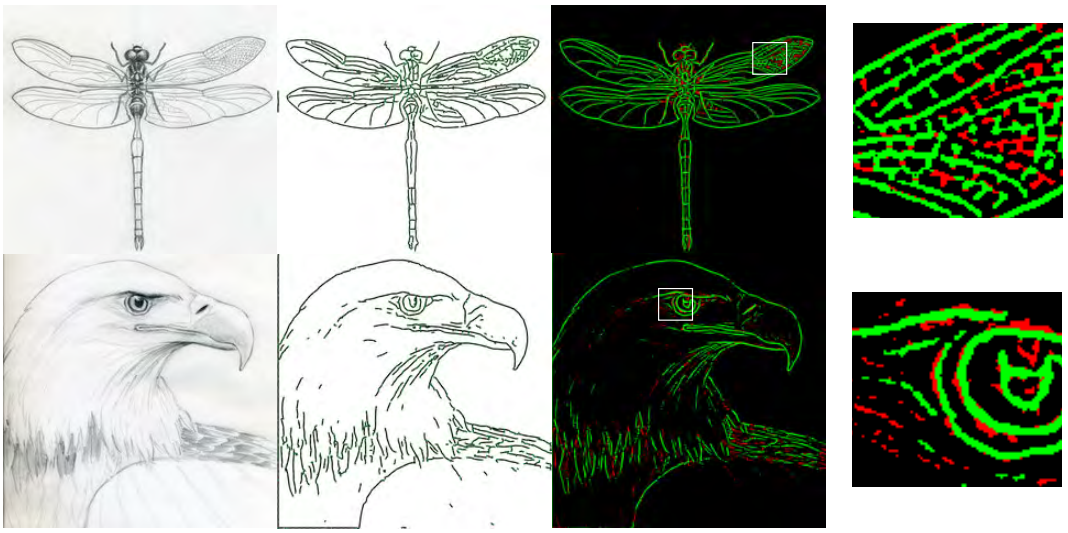

Figure 11: *Computing the accuracy of the method. The comparison is designed to examine the percentage of coverage of the second derivative determined from the image by the vector form. In the first column, we have the original image, then the vector representation and the coverage. The pixels marked in green indicate the covered surface and the red ones are not covered.*

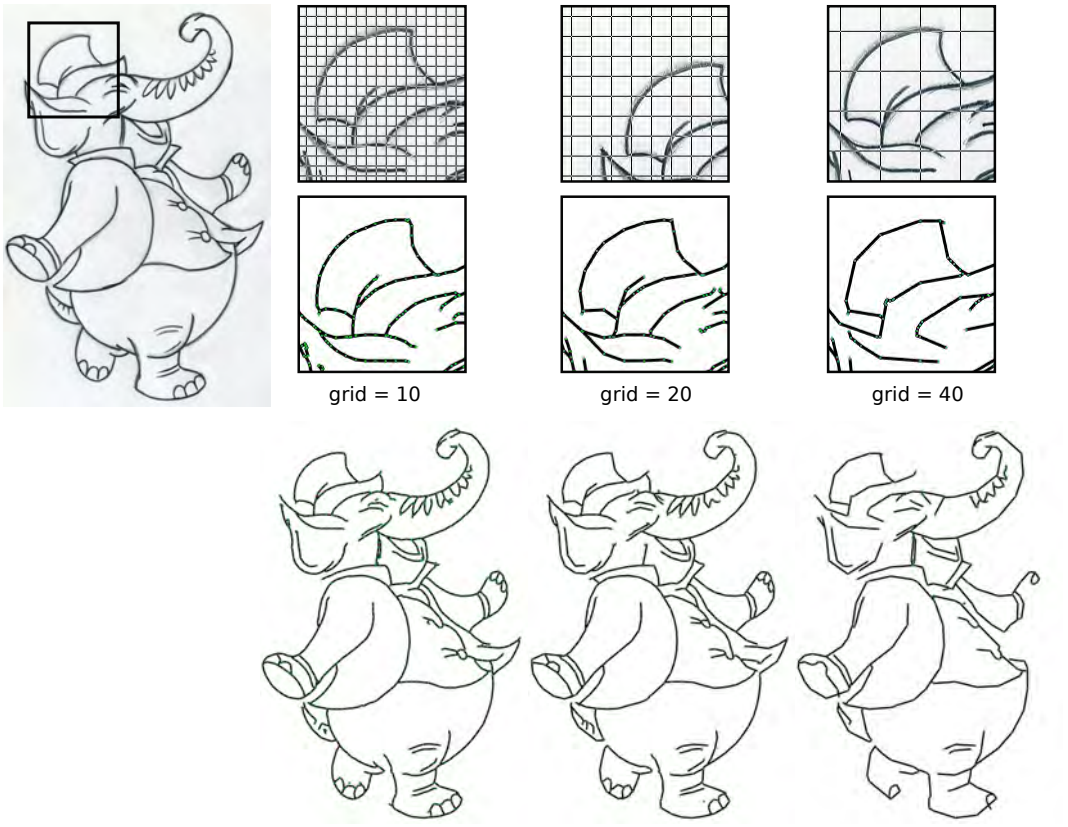

Figure 12: *Vectorization results depending on the grid size change. Experiments were conducted for 10-, 20- and 40-pixel grids. Increasing the grid size reduces the number of nodes (green dots) per line but also increases detail loss.*

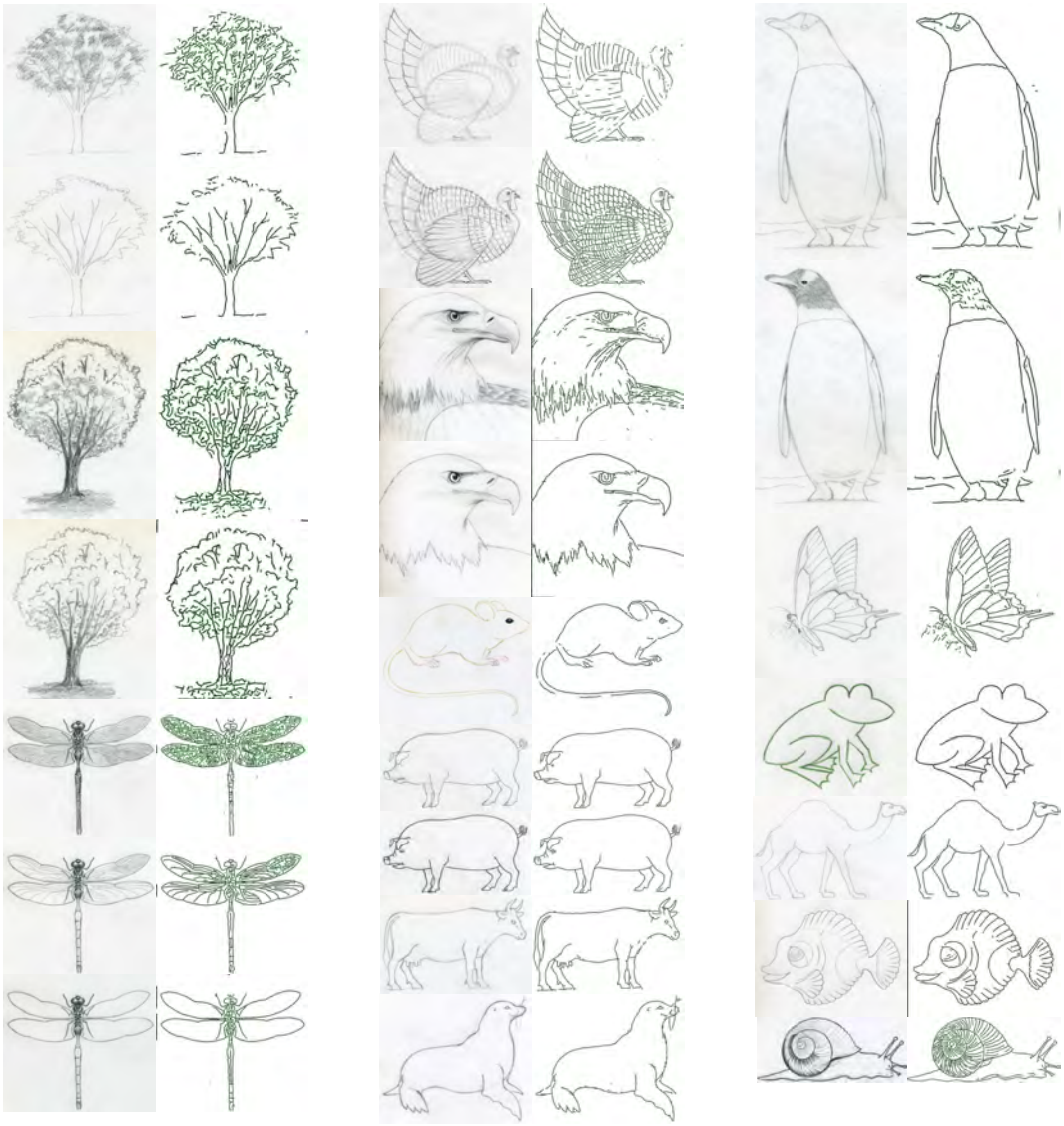

Figure 13: *Several additional examples of the experimental results.*

Figure 14: *Several additional examples of the experimental results on the CorelDraw clipart collection.*