Technical Report

Pattern Recognition and Image Processing Group
Institute of Visual Computing and Human-Centered Technology
TU Wien
Favoritenstrasse 9-11/193-03
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801 - 18661
Fax: +43 (1) 58801 - 18697
E-mail: mlanger@prip.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

PRIP-TR-147 June 2, 2019

Elliptical Line Voronoi Skeletons on the GPU

Maximilian Langer

Abstract

Calculation of a shape skeleton can be slow, therefore an implementation on the GPU is beneficial. This report covers the algorithm and implementation of an Elliptical Voronoi Skeleton and compares implementations on the CPU and GPU. Additionally, a speed improved CPU version using Python's Numba library is compared. An advanced residual calculation on a per pixel level is introduced and theoretical, although computationally expensive, elliptical residual calculations are given. A short comparision of Elliptical Voronoi and Classical Line Voronoi is given.

1 Introduction

Shape skeletons are a widely used technique in computer vision. There exist many different methods of calculating the skeleton or medial axis of a shape. In a previous work [4] we have introduced an algorithm that makes use of distance maps to calculate off-centered skeletons of polygonal shape approximations. This algorithm allows for arbitrary distance maps. The paper has a focus on elliptical distance, calculating Elliptical Line Voronoi Skeletons.

Here we want to extend this algorithm by improving on the skeleton candidate selection and give experimental results on three implementations with different speed gains.

The remainder of the report is organized as follows: Section 2 gives an overview of Elliptical Line Voronoi Skeletons. Section 3 introduces improvements to Elliptical Line Voronoi Skeletons and the adaptation of the algorithm. Section 4 explains implementation details with Python. Section 5 presents the results of the different implementations and Section 6 concludes the report.

2 Elliptical Line Voronoi Skeletons

In theory the proposed Algorithm works for arbitrary distances, but was constructed with Elliptical Line Voronoi Skeletons (ELVD Skeletons) in mind. In this section a short introduction to Elliptical Line Voronoi Skeletons is given.

2.1 Confocal Elliptical Distance

Gabdulkhakova and Kropatsch [2] introduced a point to line segment distance $d_e(P, l)$ based on confocal ellipses that defines the distance of a point $P \in \mathbf{R}^2$ to the line segment l between points $F_1, F_2 \in \mathbf{R}^2$, the *Confocal Elliptical Distance (CED)*. F_1, F_2 are the focal point of an ellipse that includes P in its boundary. The distance is given by

$$d_e(P, l) = \delta(P, F_1) + \delta(P, F_2) - \delta(F_1, F_2) \tag{1}$$

where δ denotes the Euclidean distance.

Compared to the Hausdorff distance $d_h(P, l) = \min_{L \in l} \delta(P, L)$ the CED does only take endpoints of a line segment into account.

2.2 Elliptical Line Voronoi Diagram

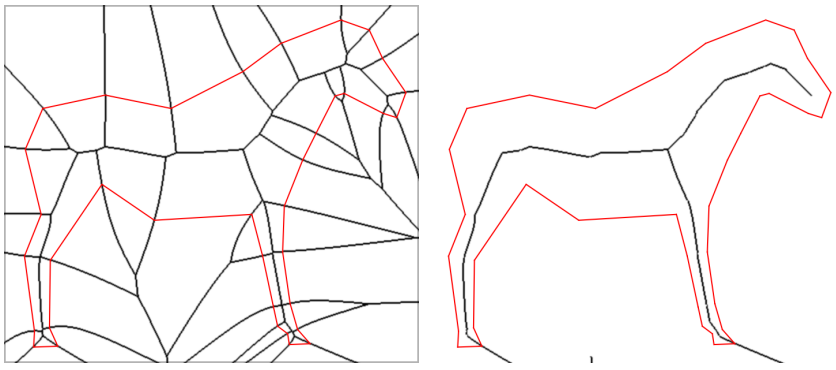
Based on the CED Gabdulkhakova et al. [3] introduced *Elliptical Line Voronoi Diagrams (ELVD)*. Similar to *Line Voronoi Diagrams (LVD)* it partitions the plane into regions where all points within are closest to a particular line (=site). LVD uses Hausdorff distance, ELVD uses CED. The boundaries between Voronoi regions are called *Voronoi edges*.

2.3 Line Voronoi Skeletons

Ogniewicz and Ilg [6] introduced *Voronoi Skeletons (VS)* that calculate the skeleton of an image shape by building a Point Voronoi Diagram. The boundary points are the sites of the Voronoi Diagram, resulting in skeletons where skeleton edges go through pixel connections. By using lines Voronoi edges go through pixels instead and the skeleton can also be calculated on a polygonal approximation. Mayya and Rajau [5] introduces Line Voronoi Skeletons with a simple pruning strategy to eliminate some spurious branches.

2.4 Elliptical Line Voronoi Skeletons

Building on ELVDs and LVS Langer et al. [4] introduced *Elliptical Line Voronoi Skeletons* with a pruning strategy based on the one of Ogniewicz and Ilg. Again they use CED instead of Hausdorff distance for Skeleton creation. Figure 1 gives an example Elliptical Line Voronoi Skeleton and the ELVD it was constructed from.



(a) Horse ELVD. (b) Skeleton generated from (a)

Figure 1: ELVD and constructed skeleton using method from [4]

2.5 Difference between ELVD and LVD

Major differences between ELVD and LVD were already highlighted in the work by Gabdulkhakova et al. [3], but there exist another property concerning the boundaries between Voronoi cells. Most applications using LVDs split a line site into three separate sites: an open line segment site and two point sites. This is done to avoid boundaries between cells that consist of two-dimensional surfaces. ELVD on the other hand does not introduce those surfaces. See Fig. 2 for an illustration.

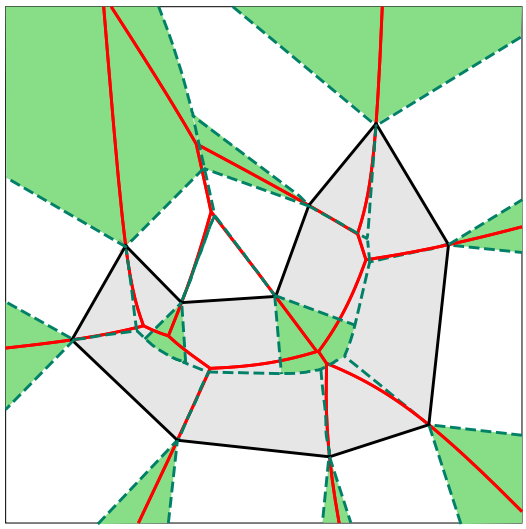


Figure 2: ELVD (red) and LVD (green dashed) of a polygon. Points in the green area are equidistant to two sites.

We have conducted an experiment on the MPEG-7 shape dataset [1] where we do not split the line sites and ended up with an average of 5% boundary pixels (this includes line and surface boundaries). On the other hand, the Elliptical Line Voronoi Diagramm does not have this problem and has an average of 0.04% boundary pixels on the above experiment. Although not as important for skeletonization, because convex Voronoi boundaries are omitted anyway we still consider this a good property of the Elliptical Line Voronoi Diagram.

[1] `http://www.dabi.temple.edu/~shape/MPEG7/dataset.html`

3 Improved Residual Calculations

Ogniewicz [7] proposes four different residual calculations to decide whether a Voronoi edge belongs to the skeleton or not. Since his method operates on point sides Voronoi edges that build up the skeleton are rather short. For LVD and ELVD they tend to be longer and therefore removing whole Voronoi edges introduces errors. In this section an improved pixel-wise calculation of the four residuals is given.

3.1 Different Residuals

For all residuals the anchor points of the Voronoi edge must be known. For the original Voronoi skeletons they correspond to the sites that generate the edge, for the Line Voronoi Skeletons proposed before [4] the midpoints of the line sites are used. In this work the anchor points are the closest points on the line segments to the pixel in consideration.

The simplest residual is the *Potential Residual* ΔR_P which is the distance along the boundary between the two anchor points. The *Chord Residual* ΔR_H subtracts the Euclidean distance between the anchor points from the Potential Residual. The *Circular Residual* subtracts the smaller arc length of the arcs centered at the Voronoi edge/pixel in question and spanning between the two anchor points from the Potential Residual. The *Bicircular Residual* is similar to te Circular Residual but multiplies the Potential Residual by $\frac{\pi}{2}$ before subtraction and scaling the result by $\frac{2}{\pi}$ to be comparable to the other residuals. See Fig. 3

3.2 Anchor points in LVD and ELVD

As stated above, the calculation of anchor points is important. For point site Voronoi diagrams the anchor points are given as the point site coordinates. For LVD the anchor points correspond to the closest point on the line site to the point P on the Voronoi edge in question.

For ELVD the anchor points A_1, A_2 correspond to the intersection points of hyperbola arms h_1, h_2 through P with foci in two opposite points (see Fig. 4) of the two line sites ((A, B), (C, D)) and the line sites themselves, resulting in foci A, D and C, B. On each line the closer point is chosen.

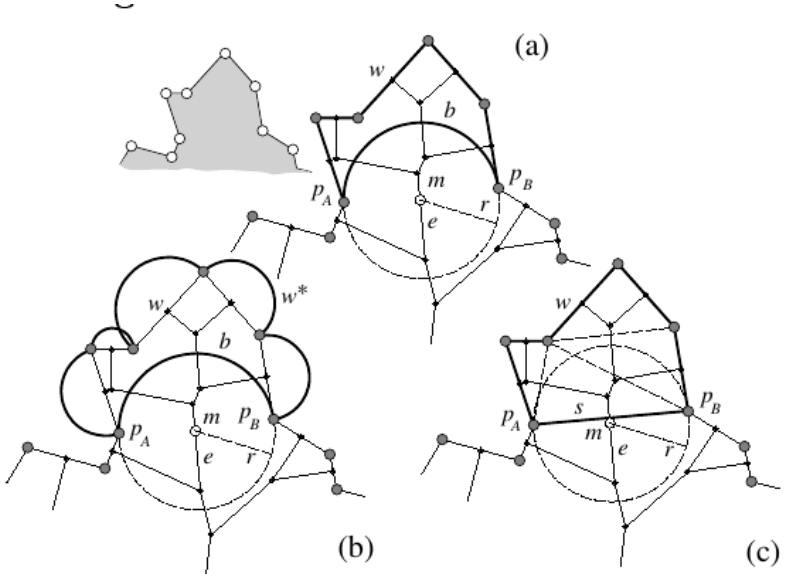


Figure 3: Different Residuals: (a) Circular Residual, (b) Bicircular Residual and (c) Chord Residual. Figure from [6]

3.3 Circular Residual

The Circular Residual, as well as the Bicircular Residual, use the circle arc length between the two anchor points. For LVD this is straight forward but for ELVD it is more complicated. In the following, two approaches are discussed: Elliptical arc using a midpoint on the LVD and an approximated circle arc.

3.3.1 Elliptical Arc

As we use elliptical Voronois we can use an elliptical arc that better approximates the boundary in elongated shapes. See Fig. 5 for an illustration of the following explanation. After obtaining the anchor points as described above, the intersection point M of the two tangents (dashed lines) of the before mentioned hyperbolic branch (see Fig. 4) at the anchor points A_1, A_2 is chosen as midpoint T. This point lies on the LVD if all four intersection points of the hyperbolic branches h_1, h_2 with the line segments exist. Furthermore the endpoints E_1, E_2 of the LVD edge are needed to construct two ellipses that have one focal point F_1 in those endponints. The other focus F_2 can be constructed by mirroring the endpoint about the tangent used above

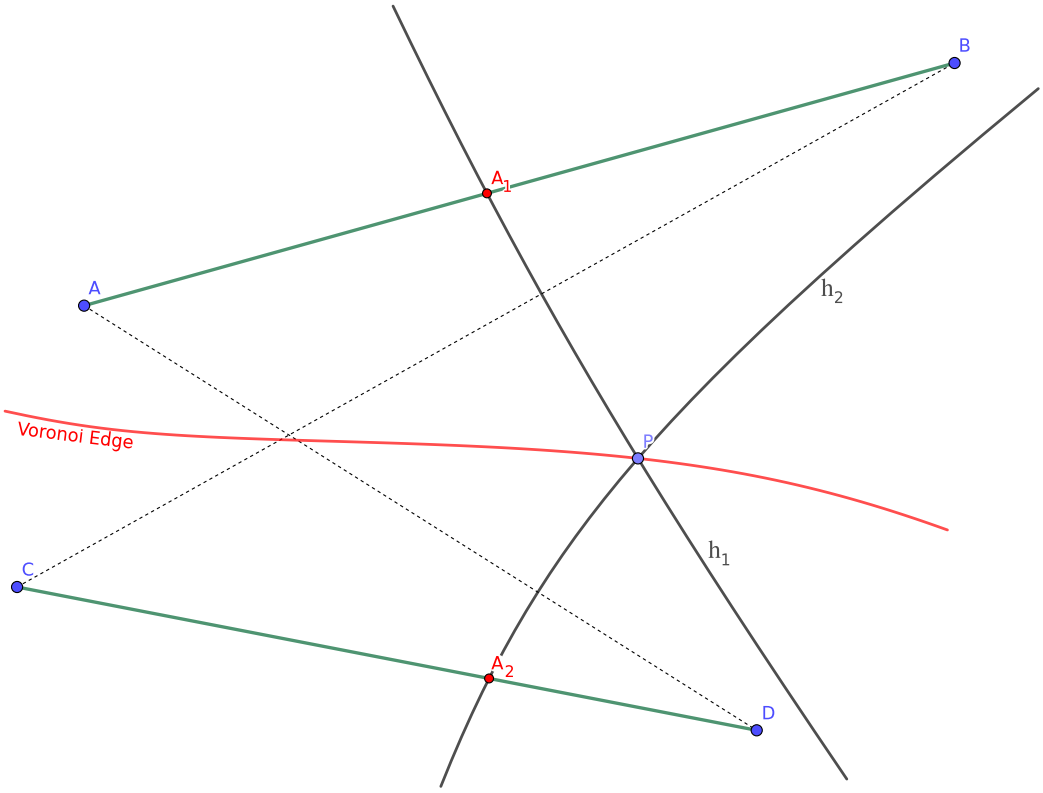


Figure 4: Anchor points of ELVD. Dotted lines are the hyperbola major axises.

into a temporary point T_1, T_2. The intersection point of a line constructed by T_1/T_2 and the anchor point of the used tangent with the line constructed by the endpoints is the second focus point F_2. There exists an ellipse that includes both anchor points. After constructing the two ellipses one can then create one arc for each ellipse that goes around the focal point that coincides with the endpoint. The smaller one of those two arcs can then be used for residual calculations.

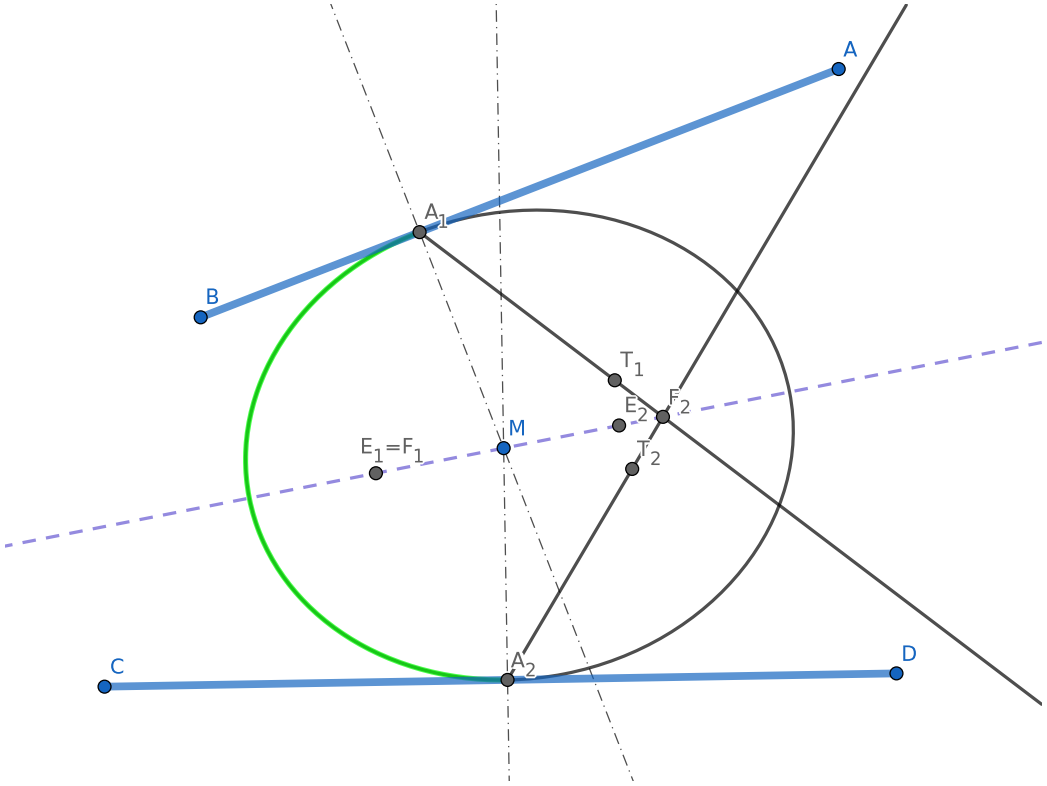


Figure 5: Construction of elliptical arc (green) from ellipse with foci F_1,F_2.

3.3.2 Approximated Circle Arc

As the goal of this project is to speed up ELVD skeleton calculation the approach above is not ideal. It is faster to calculate the circle arc length as given here.

First the anchor points are constructed by looking for the closest (Euclidean distance) point on the line sites. If the closest point does not lie on

the line segment (site) and the two constructing sites are connected then this Voronoi edge point is leading to a convex part and can be dismissed, i.e. it is not a valid skeleton point. After calculating the anchor points, the circular arc radius is determined by taking the mean distance of the two anchor points to the Voronoi edge point. The Arc length is then calculated likewise to the LVD.

4 Implementation in Python

The following section gives some details on the implementation in Python. Three different methods were implemented and tested (see Sect. 5). The three methods are a native Numpy Python implementation, a CPU enhanced Numba implementation and a GPU version again using Numba with CUDA.

4.1 NumPy and Numba

Python is a high level programming language and provides, together with NumPy, a scientific computing language. NumPy extends the basic language with powerful nD array modification and scientific tools that are implemented in C/C++ and therefore run very fast with a feature set that is compatible to other modern solutions (like e.g. MATLAB).

To further enhance the speed of NumPy and Python code, Numba can be used. It utilizes a JIT (just-in-time) compiler to run specific code parts in C/C++ speed on CPU. For experiments in this project it is interesting that Numba also allows the creation of JIT function that are compiled for GPU using Nvidia's CUDA library. This allows to build GPU executable code from within Python.

The combination of NumPy and Numba leaves three different approaches for the algorithm: Basic NumPy, NumPy with time critical functions JIT compiled and with those functions run on the GPU. Note that a naive function without NumPy that calculates values per pixel on the CPU was not implemented as it would lead to very poor performance.

4.2 The Algorithm

The algorithm was implemented three times. Version 1 (V1) is implemented only on the CPU, Version 2 (V2) is partly implemented with Numba's JIT

functions and Version 3 (V3) is mostly implemented on the GPU (using Numba). Figure 6 gives an overview of the steps of the algorithm. First the segmented image is approximated as a polygonal shape as needed for ELVD. Then a distance map for a single point is created. This is later used in the ID map creation together with the polygon. The ID map gives every pixel the ID of the closest line segment of the polygon. Then the Boundary Maps are extracted. There are two boundary maps, a small and a big map. Boundary maps are undefined for non-boundary pixels and the small boundary map saves the lower ID and the big boundary map the bigger ID of a Voronoi edge. All pixels that have a different valued neighbor in horizontal or vertical direction in the ID map belong to a Voronoi edge. For every valid pixel in the boundary maps (note that if a pixel is valid in one boundary map it is valid in the other one too) a residual value is calculated. In a last step only boundary pixels with a residual value higher than a certain threshold are included in the skeleton.

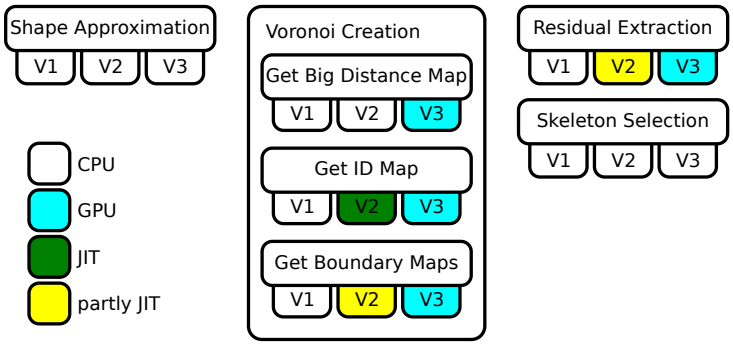


Figure 6: Overview of the algorithm and where the computation is done.

The algorithm was first implemented on the CPU only. After benchmarking different parts of the algorithm, slow parts were implemented using Numba's JIT in V2. Finally, all grid based steps of the algorithm were implemented on the GPU (V3). Note that for the sake of comparability the steps were kept similar to the CPU version. Faster implementations with everything implemented on the GPU are possible.

5 Results

In this section a time comparison is presented based on multiple tests. Also some example skeletons are shown.

5.1 Experiment Configurations

The three versions of the algorithms were tested on the MPEG-7 shape dataset[2]. The dataset consists of 1400 images in various sizes ranging from around 2500 total pixels to around 1 million total pixel, that is around 50×50 to 1000×1000. In total 12 runs were executed. For each run three values were varied:

Approximation Tolerance The polygon is approximated from the original pixel shape with the Douglas-Peucker algorithm [1]. This gives the tolerance level. Higher tolerance yields fewer lines in polygon.

Image Scale This is the scaling factor the image is scaled before processing.

Polygon Scale This gives the scaling factor for scaling the polygon (and resulting skeleton image).

As comparison with a state-of-the-art, parallel skeletonization algorithm (ZS84) was done [8]. Note that this algorithm produces different results, as it does a thinning approach, not an elliptical skeleton on a polygonal approximation of the shape. Polygon Scale is also an image scale for ZS84. Table 1 shows the different runs and their different values for approximation, image and polygon scale.

5.2 Results on CPU/GPU

All tests were conducted on a Linux Mint desktop with an AMD Ryzen 7 2700X with 8 cores at 3.70 GHz, 32 GB of memory and an NVIDIA GeForce GTX 1060 with 6GB GRAM.

Table 2 gives the average times per run and the standard deviation in parenthesis. Note that the MPEG7 dataset has different image and shape

[2] `http://www.dabi.temple.edu/~shape/MPEG7/dataset.html`

[3] Only 300/1400 images run.

Table 1: Overview of conducted test runs.

Run #	Approx. Tol.	Image Scale	Polygon Scale
Run 1	3	0.5	0.5
Run 2	3	1	0.5
Run 3	3	1	1
Run 4	3	1	3
Run 5	3	1	5
Run 6	3	3	1
Run 7	3	5	1
Run 8	5	1	3
Run 9	5	3	1
Run 10	5	1	5
Run 11	5	5	1
Run 12[3]	3	3	3

sizes. Table 3 shows the speed improvement of the Numba and GPU implementation compared to the CPU implementation. It also shows the degradation/improvement compared to ZS84. Figure 7 illustrates the values of Table 3.

As can be seen, GPU computation gets significantly faster with bigger images, even gaining vast speedups compared to the very performant ZS84 C++ implementation (using SkImage [4]). The Numba implementation on the other hand is not much faster than the default Numpy implementation. This most likely comes from the fact that Numpy already does a lot of optimization and Numba needs to compile the code first (Note that this is only done once for the run of 1400 images) without significant improvement later on.

5.3 Example Skeletons

Figure 8 gives six examples from the MPEG-7 dataset and the ELVD skeletons generated by the GPU. Note in Fig 8f the skeleton leaves the object. This is because of shape approximation for line generation needed for ELVD. All example images use the Circular Residual.

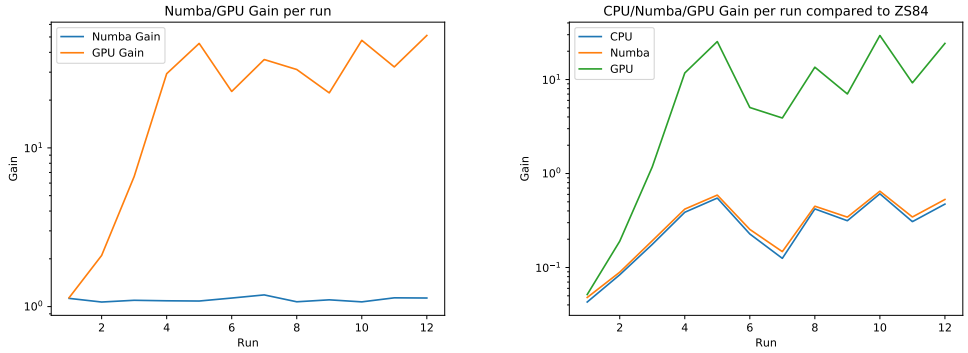
[4] `http://scikit-image.org/docs/dev/api/skimage.html`

Table 2: Average times of algorithm on MPEG7 dataset in ms.

Run #	ZS84	CPU	Numba	GPU
Run 1	0.42 (0.00043)	9.5 (0.00674)	9.94 (0.05155)	8.23 (0.0198)
Run 2	2.67 (0.00326)	31.57 (0.03565)	29.86 (0.03304)	13.09 (0.00818)
Run 3	19.24 (0.02433)	126.9 (0.17516)	114.41 (0.1539)	15.26 (0.01161)
Run 4	497.74 (0.66767)	1404.94 (2.07993)	1266.85 (1.81363)	42.23 (0.05074)
Run 5	2338.03 (3.11432)	5115.79 (7.44559)	4567.13 (6.41103)	100.35 (0.13695)
Run 6	499.51 (0.67034)	3925.68 (10.94507)	3338.29 (9.17605)	124.87 (0.20994)
Run 7	2320.01 (3.090.82)	39749.41 (77.78169)	33108.99 (64.78515)	761.55 (1.24783)
Run 8	495.98 (0.66541)	1209.6 (1.64169)	1109.98 (1.44618)	33.89 (0.03618)
Run 9	498.98 (0.66918)	2102.86 (4.34114)	1843.41 (3.67406)	74.88 (0.10188)
Run 10	2322.49 (3.08909)	4134.24 (5.34866)	3761.56 (4.68866)	78.94 (0.09765)
Run 11	2326.2 (3.10157)	15467.9 (34.55211)	13117.13 (28.92259)	344.55 (0.63329)
Run 12	13726.5 (19.17575)	48014.31 (97.04433)	41494.66 (83.55067)	927.57 (1.95543)

Table 3: Average gains if using Numab/GPU algorithm (first and second values) and average gain to ZS84 algorithm using CPU and GPU algorithm.

Run #	Numba Gain	GPU Gain	CPU Gain to ZS84	GPU Gain to ZS84
Run 1	1.13 (0.06)	1.14 (0.32)	0.04 (0.03)	0.05 (0.04)
Run 2	1.07 (0.06)	2.1 (0.87)	0.08 (0.06)	0.19 (0.19)
Run 3	1.1 (0.05)	6.6 (3.69)	0.18 (0.13)	1.18 (1.18)
Run 4	1.09 (0.04)	29.34 (9.2)	0.39 (0.28)	11.74 (11.01)
Run 5	1.08 (0.05)	45.54 (12.26)	0.55 (0.41)	25.35 (22.3)
Run 6	1.13 (0.05)	22.67 (7.8)	0.23 (0.2)	5.04 (5.06)
Run 7	1.18 (0.04)	36.03 (14.96)	0.13 (0.1)	3.9 (2.89)
Run 8	1.07 (0.04)	31.18 (10.07)	0.42 (0.29)	13.53 (12.11)
Run 9	1.1 (0.05)	22.21 (7.08)	0.31 (0.25)	7.02 (6.57)
Run 10	1.07 (0.04)	47.62 (12.35)	0.61 (0.43)	29.43 (24.3)
Run 11	1.14 (0.05)	32.36 (10.85)	0.31 (0.27)	9.22 (8.24)
Run 12	1.13 (0.04)	51.09 (9.67)	0.47 (0.44)	24.23 (24.54)



(a) Gain of Numba/GPU compared to CPU implementation.

(b) Gain of CPU/Numba/GPU compared to ZS84.

Figure 7: Speed Relation of different algorithms.

6 Conclusion and Future Work

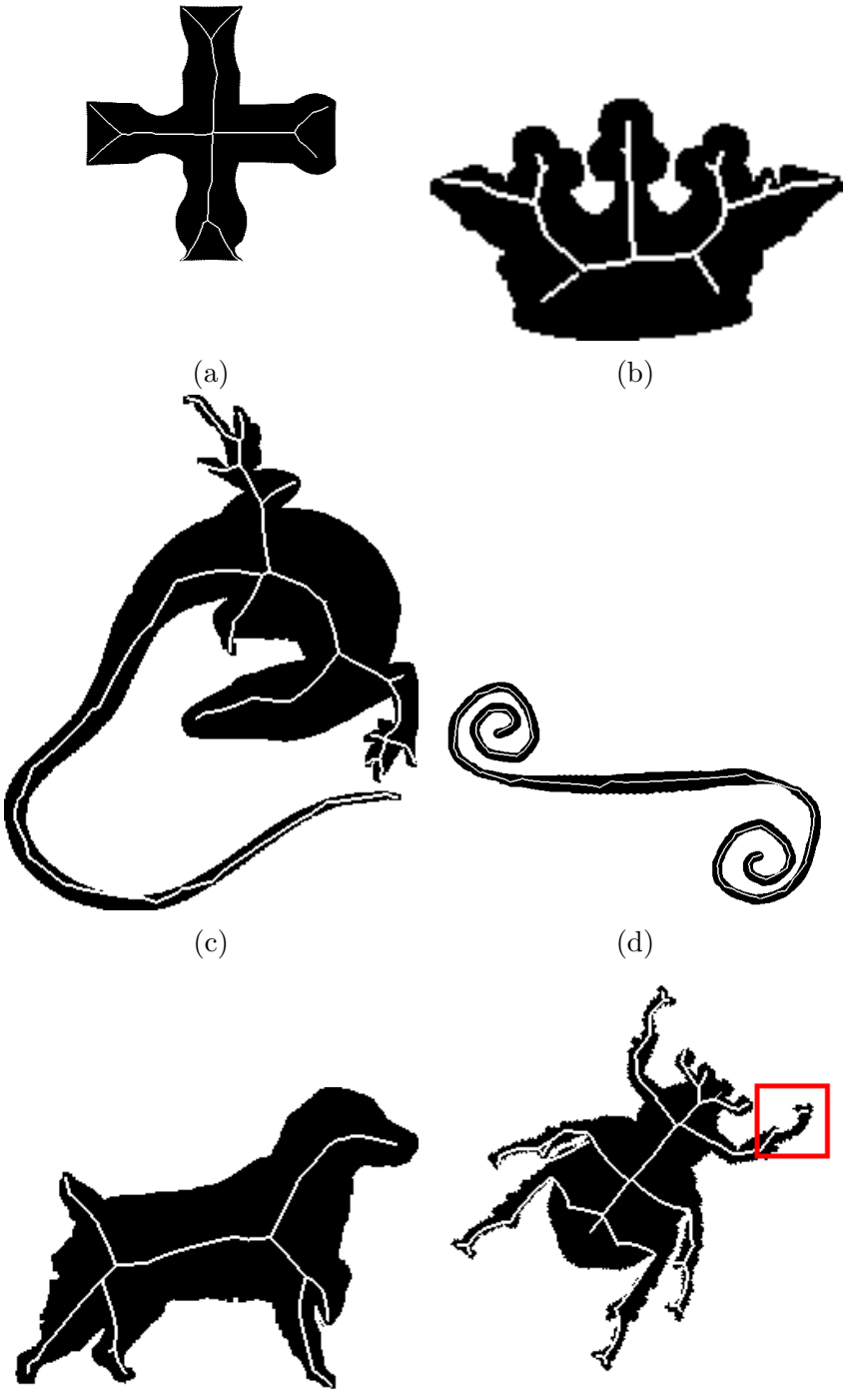
This work presents an improved Elliptical Voronoi Skeleton and goes into detail of implementation on CPU and GPU. The provided results show that even for big images ($> 5000 \times 5000$ pixels) an implementation on GPU hardware can lead to fast computation.

GPU implementation is already promising in the way presented here, but could be improved in various ways: all steps on GPU; dedicated shader or smart memory management (3D arrays). Furthermore, Elliptical Voronoi Skeletons must be further researched and their application domain established. We now know that efficient computation is possible.

References

[1] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[2] Aysylu Gabdulkhakova and Walter. G. Kropatsch. Confocal ellipse-based distance and confocal elliptical field for polygonal shapes. In *Proceedings of the 24th International Conference on Pattern Recognition, ICPR*, 2018.



(a) (b) (c) (d) (e) (f)

Figure 8: Example skeletons. Note that in (f) there are errors because of the approximation of the boundary.

[3] Aysylu Gabdulkhakova, Maximilian Langer, Bernhard W Langer, and Walter G Kropatsch. Line voronoi diagrams using elliptical distances. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, 2018.

[4] Maximilian Langer, Aysylu Gabdulkhakova, and Walter G Kropatsch. Non-centered voronoi skeletons. In *International Conference on Discrete Geometry for Computer Imagery*, pages 355–366. Springer, 2019.

[5] Niranjan Mayya and VT Rajan. Voronoi diagrams of polygons: A framework for shape representation. *Journal of Mathematical Imaging and Vision*, 1996.

[6] R Ogniewicz and Markus Ilg. Voronoi skeletons: Theory and applications. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, 1992.

[7] Robert L Ogniewicz. *Discrete voronoi skeletons*. PhD thesis, ETH Zurich, 1992.

[8] TY Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.