computing@computingonline.net
www.computingonline.net

Print ISSN 1727-6209
On-line ISSN 2312-5381
International Journal of Computing

AN OPTIMIZED ALGORITHM FOR COMPUTING THE VORONOI SKELETON

Dmytro Kotsur, Vasyl Tereshchenko

Taras Shevchenko National University of Kyiv, 4d Academician Glushkov Avenue, Kyiv, Ukraine, 03680
dkotsur@gmail.com, vtereshch@gmail.com

Paper history:
Received 26 February 2019
Received in revised form 10 May 2019
Accepted 25 May 2020
Available online 30 December 2020

Keywords:
Voronoi diagram;
Voronoi graph;
skeleton;
polygon;
shape simplification;
heuristic;
optimization.

Abstract: The skeleton-based representation is widely used in such fields as computer graphics, computer vision and image processing. Therefore, efficient algorithms for computing planar skeletons are of high relevance. In this paper, we propose an optimized algorithm for computing the Voronoi skeleton of a planar object with holes, which is represented by a set of polygons. Such skeleton allows us to use efficiently the properties of the underlying Voronoi diagram data structure. It was shown that the complexity of the proposed Voronoi-based skeletonization algorithm is $O(N \log N)$, where N is the number of polygon's vertices. We have also proposed theoretically justified optimization heuristic based on polygon/polyline simplification algorithms. In order to evaluate and prove the efficiency of the heuristic, a series of computational experiments were conducted involving the polygons obtained from MPEG 7 CE-Shape-1 dataset. We have measured the execution time of the skeletonization algorithm, computational overheads related to the introduced heuristics and also the influence of the heuristic onto accuracy of the resulting skeleton. As a result, we have established the criteria, which allow us to choose the optimal heuristics for our skeletonization algorithm depending on the system's requirements.

1. INTRODUCTION

The skeletal representation of the planar object is essential for many problems of computer vision and pattern recognition, computer graphics and visualization [1]. For example, skeletons are widely used for shape matching [2, 3], optical character recognition [4] and image retrieval [2, 5]. In the biological image processing, skeletonization methods are extensively applied to extract the central line of thin objects. For example, based on the image data one can obtain the skeletal graph representing the retinal blood vessels topology [6, 7]. Similarly, this technique is applied for segmenting the biological neurons [8] and for extracting thin subcellular structures [9, 10] based on microscopy data.

Related work. There are different skeletonization methods depending on a type of input data. Morphological thinning techniques are extensively used for computing the skeleton of a binary image [11-13]. They allow us to obtain a pixel-based representation of the skeleton. In order to obtain the topology of the underlying graph, graph vectorization methods can be applied [14, 15].

Another type of techniques is based on central line tracing. They are commonly used to segment thin-structures depicted on an image (e.g., axons and dendrites of neurons [16], blood vessels [17], filamentous structures [17, 18]). These methods can directly represent the skeleton as a connected graph. However, due to an iterative nature of these methods, the execution time may vary significantly. A different type of methods is used to compute a skeleton of an object, whose shape is represented by polygonal contours (with holes). Such contours are extracted from a binary image using tracing techniques (e.g., Marching squares [19]). Alternatively, these methods can compute a skeleton of primitives obtained from a vector graphic representation.

Methods to construct the straight skeleton based on shrinking technique with $O(N \log N)$ complexity are described in papers [20, 21]. A linear complexity

method for a simple polygon without holes was introduced in [22].

Another approach for constructing the skeleton of an object with polygonal representation is by using the Voronoi diagram [23, 24]. This class of methods allows us to obtain a graph representation of the skeleton directly and use the advantages of the Voronoi diagram data structure allowing for solutions of many related problems [25] (e.g., finding a convex hull, nearest neighbor, maximal inscribed disk). However, the main drawback of such methods is high computational costs due to processing the vast number of simple primitives (points, line segments).

Therefore, in this paper, we propose an approach to decrease the computational costs related to the Voronoi-based skeletonization algorithms by introducing the novel preprocessing step based on shape simplification heuristics. Our primary focus is on investigation the suitable optimization heuristics, empirical validation of such heuristics and determining their properties.

The paper is organized as follows: The first section gives an overview of the problem and related state-of-the-art literature. In the second section, we outline the theoretical background and formulate the skeletonization problem. The third section is devoted to the description of the skeletonization algorithm, its main steps, and vertex/edge labeling procedure. In Section 4 we introduce theoretical background related to the optimization heuristics, we formulate the shape simplification requirement and analyze suitable shape simplification algorithms. In Section 5 we show the main results of our empirical study of the optimized skeletonization algorithm; we illustrate the computational overheads related to optimization heuristics and demonstrate the algorithm's performance improvements. Section 6 is devoted to the analysis of the obtained results. It establishes the criteria allowing the selection of the optimal heuristics for Voronoi-based skeletonization algorithm depending on the system's requirements. Finally, the paper concludes with a summary of the main results.

2. PROBLEM STATEMENT

We assume that the planar object has continuously-differentiable boundaries (called G_1-continuous) except for a finite number of *critical points*, where contour is continuous, but not differentiable (G_0-continuous points). The object's boundaries are represented by a set of simple polygons $\mathcal{S} := \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m\}$, where polygon $\mathcal{P}_k$ is defined as an ordered set of points $p_1^k, p_2^k, \dots, p_{M_k}^k$.

The set of open line segments corresponding to the polygon $\mathcal{P}_k$ is denoted by $\mathcal{L}_k := \mathcal{L}(\mathcal{P}_k) = \left\{l_i^k := \left(p_i^k, p_{i+1}^k\right) \mid i = 1, \dots, M_k, p_{M_k+1}^k = p_1^k\right\}$. The set of all vertices of the polygon (line segment's endpoints) is denoted by $\mathcal{Q} = \bigcup_{k=0}^{m} \bigcup_{j=1}^{M_k} \{p_i^k\}$.

Let us denote by $\mathcal{L} := \bigcup_{k=1}^{m} \mathcal{L}_k$ the set of all line segments representing the planar object. We also assume that the polygon $\mathcal{P}_0$ represents the outer contour of the object and polygons $\mathcal{P}_1, \dots \mathcal{P}_m$ correspond to the inner holes. Thus, $R := \mathcal{P}_0 \setminus \bigcup_{i=1}^{m} \mathcal{P}_i$ defines the object's region (domain).

Definition 1. The *Voronoi cell* corresponding to an element $u \in \mathcal{L} \cup \mathcal{Q}$ is a locus of points:

$$\mathcal{VC}(u) = \{p \in \mathbb{R}^2 \mid \|p - u\| \le \|p - w\|, w \ne u, w \in \mathcal{L} \cup \mathcal{Q}\} \quad (1)$$

Definition 2. The *Voronoi diagram* of a set of line segments $\mathcal{L}$ (with endpoints $\mathcal{Q}$) is defined as a set of all Voronoi cells:

$$\mathcal{VD}(\mathcal{L}, \mathcal{Q}) = \bigcup_{u \in \mathcal{L} \cup \mathcal{Q}} \{\mathcal{VC}(u)\} \quad (2)$$

Remark 1. Note that the *Voronoi diagram* of line segments $\mathcal{L}$ (with endpoints $\mathcal{Q}$) is defined by the boundaries between the neighboring Voronoi cells. The most of the computational algorithms (e.g., "Divide and Conquer" [26], Fortune's algorithm [27]) represent such boundaries in terms of the Voronoi graph [28-29] $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ with a set of the Voronoi vertices $V_{\mathcal{S}}$ and a set of Voronoi edges $E_{\mathcal{S}} \subseteq V_{\mathcal{S}} \times V_{\mathcal{S}}$.

Definition 3. A finite set of polygon's $\mathcal{P}$ points, where the object is G_0-continouous (but not G_1-continouous) are called *critical points (vertices)* of the polygon $\mathcal{P}$.

Remark 2. Note that the vertices of the polygon $\mathcal{P}$, which correspond to G_1-continouous part of the object's boundary, induce redundant edges of the Voronoi diagram – the bisectors between two consecutive line segments l_i and l_{i+1}, which share a common *non-critical* vertex p_i. In order to obtain an approximate Voronoi diagram of an object represented by $\mathcal{S}$, such redundant edges corresponding to all *non-critical* points of $\mathcal{S}$ should be removed.

Definition 4. The *approximate Voronoi diagram* $\mathcal{VD}_a(\mathcal{S})$ for a planar object represented by a set of polygons $\mathcal{S}$ is obtained as a subgraph $G_{\mathcal{S}}^a$ of the Voronoi graph $G_{\mathcal{S}}$ obtained by removing the edges of $G_{\mathcal{S}}$ corresponding to the bisectors between two

consecutive line segments l_i and l_{i+1}, which share a common *non-critical* vertex p_i.

Definition 5. The *Voronoi skeleton* of a planar object represented by a set of polygons $\mathcal{S}$ is a subset of the *approximate Voronoi diagram* $\mathcal{VD}_a(\mathcal{S})$ positioning inside the object's region R.

Remark 3. Thus, the *Voronoi skeleton* of $\mathcal{S}$ is obtained by removing (or trimming) the edges of $G_{\mathcal{S}}^a$, which do not belong to R

Problem statement: Given a set of polygons $\mathcal{S}$, which represent a planar object, construct the Voronoi skeleton of $\mathcal{S}$.

3. SKELETONIZATION METHOD

In this section, we provide a description of the Voronoi skeletonization algorithm and Voronoi graph processing routine. We also analyze the algorithm's complexity (see Subsection 3.3).

3.1 SKELETONIZATION ALGORITHM

The input of the algorithm and its main steps are the following:

Input: $\mathcal{S} := \{\mathcal{P}_1, \mathcal{P}_2, \dots, \mathcal{P}_m\}$ – the set of polygons, each vertex p_i^k of the polygon $\mathcal{P}_k$ has a bool attribute $isCritical[p_i^k] \in \{0,1\}$. Polygon $\mathcal{P}_k$ is oriented such that the interior of the object is to the right for any line segment $l \in \mathcal{P}_k$.

Algorithm:

Step 1: Construct the Voronoi diagram of a line segments $\mathcal{L}$ with endpoints $\mathcal{Q}$ (Fig. 1). Obtain the Voronoi graph $G_{\mathcal{S}} = (V_{\mathcal{S}}, E_{\mathcal{S}})$ represented as a doubly-connected edge list (DCEL);

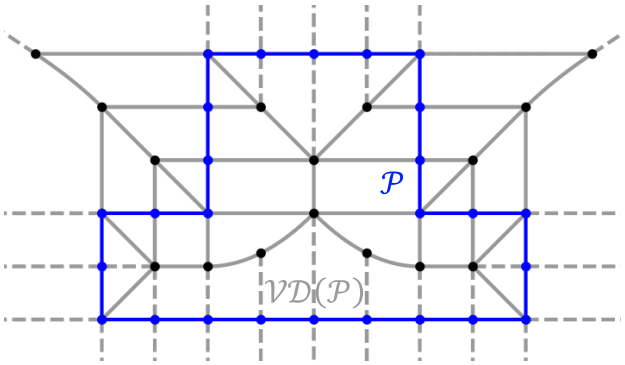

Figure 1 – Examples of the Voronoi diagram (gray, black) for polygon's (blue) edges

Step 2: Run the breadth-first search (BFS) algorithm, traverse graph $G_{\mathcal{S}}$ and label its edges and vertices (Fig. 2) as described in Subsection 3.2;

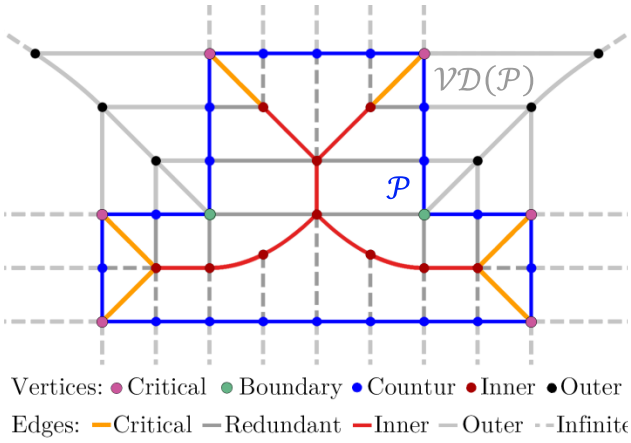

Figure 2 – Examples of the labeled Voronoi vertices and edges

Step 3: Remove all "Redundant", "Outer" edges and "Boundary", "Outer" vertices of $G_{\mathcal{S}}$ (Fig. 3);

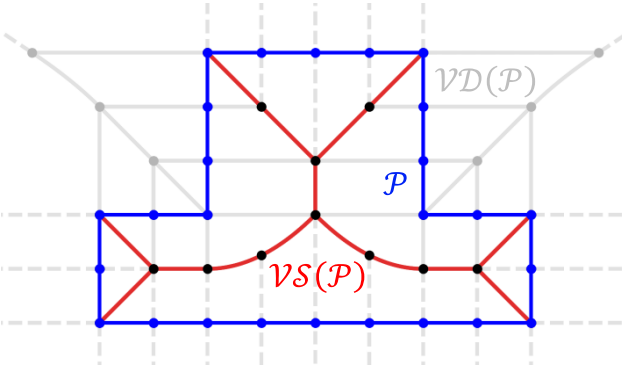

Figure 3 – Examples of the Voronoi skeleton (red lines and black vertices)

Step 4: Remove isolated vertices of $G_{\mathcal{S}}$ if any exist;

3.2 LABELING THE VORONOI GRAPH

We traverse the edges and vertices of the Voronoi graph $G_{\mathcal{S}}$ and mark them according to their role in a resulting graph of the Voronoi skeleton.

Definition 6. The Voronoi vertex is called:

- *Inner*, if it is inside the object's polygon;
- *Outer*, if it is outside the object's polygon;
- *Critical*, if it coincides with one of the critical vertices of the object's polygon;
- *Boundary*, if it coincides with one of the non-critical vertices of the polygon;

Definition 7. The Voronoi edge is called:

- *Inner*, if it is located inside object's region and doesn't touch its boundaries;
- *Critical*, if it is located inside object's region and adjacent to a critical vertex;
- *Outer*, if the edge is located outside the object's polygon;

- *Redundant*, if it is located inside object's polygon and touches polygon's non-critical vertex;

Therefore, the labels for Voronoi vertices are (see Fig. 2) *Inner* – "I" *Critical* – "C", *Outer* – "O", *Boundary* – "B"; The labels for Voronoi edges are *Inner* – "I", *Critical* – "C", *Outer* – "O", *Redundant* – "R" (see Fig. 2). The types of the Voronoi cells are *Endpoint* – "EP" and *Line segment* – "LS";

Note that *Outer* and *Boundary* vertices are absent in the final Voronoi skeleton and therefore should be removed from the Voronoi graph together with *Outer* and *Redundant* edges.

We label the edges and vertices of G_S iteratively using the breadth-first search procedure (BFS) Firstly, we initialize the queue of BFS algorithm by infinite edges of a Voronoi graph. The pseudocode for the BFS initialization procedure is shown in the listing below:

```
function InitQueue(Q):
  Q := EmptyQueue();
  for edge e do
    Label[e] = "None";
    if isInfinite(e) then
      EnQueue(e, Q);
      v := non-null vertex of e;
      Label[v] = "None";
    else
      v1, v2 := non-null vertices of e;
      Label[v1] = Label[v2] = "None";
    end;
  end;
  return Q;
end;
```

Label [•] is a data structure storing the labels of edges and vertices. Using the BFS we traverse all edges of the Voronoi graph starting from the infinite edges. At each iteration we label current edge and the adjacent non-labeled vertex. See the listing below:

```
procedure TraverseBFS(Q):
  while not Empty(Q) do
    e := DeQueue(Q);
    v := Null;
    if isInfinite(e) then
      v := non-null vertex of e;
      LabelInfinite(e); // Classify the edge and incident vertex
    else
      v := vertex of e with "None" label;
      LabelFinite(e); // Classify the edge and incident vertex
    end;
    for edge e incident to v do
      // Add non-labeled edges to queue
      if Label[e] = "None" then
        EnQueue(e, Q);
      end;
    end;
  end;
end;
```

The following function determines the label of an infinite edge and the corresponding vertex:

```
procedure LabelInfinite(e):
  c1, c2 := cells corresponding to e and Twin(e);
  v := non-null vertex of e;
  if Type(c1) = "EP" and Type(c2) = "EP" then
    Label[e] = "O"; // Outer
    Label[v] = "O"; // Outer
  else // Edge between line segment's interior and its endpoint
    p := is a unique endpoint of line segment;
    if v coincides with p then
      if isCritical[p] then
        Label[v] := "C"; // Critical
      else
        Label[v] := "B"; // Boundary
      end;
      Label[e] := "O"; // Outer
    else
      Label[v] := "I"; // Inner
      if isCritical[p] then
        Label[e] := "C"; // Critical
        Trim e to p;
      else
        Label[e] := "R"; // Redundant
      end;
    end;
end;
```

The following function determines the label of finite edges and adjacent non-labeled vertex:

```
procedure LabelFinite(e):
  v0 := labeled vertex of e;
  v1 := unlabeled vertex of e; // Label[v1] is "None"
  c1, c2 := cells corresponding to e and Twin(e);
  if Label[v0] = "I" or Label[v0] = "O" then
    if Type(c1) = "LS" and Type(c2) = "LS" then
      if line segments in c1 and c2 share endpoint p then
        if isCritical[p] then
          Label[v1] := "C"; // Critical
          Label[e] := if Label[v0] = "I" then "C" else "O";
        else
          Label[v1] := "B"; // Boundary
          Label[e] := if Label[v0] = "I" then "R" else "O";
        end;
      else
        Label[v1] := Label[v0];
        Label[e] := Label[v0];
      end;
    else // Edge between LS and EP
      if c1 and c2 belong to the same LS then
        p := line segment's endpoint; // Bisector is LS
        if p coincides with v1 then
          Label[v1] := if isCritical[p] then "C" else "B";
          if Label[v0] = "O" then
            Label[e] := "O"; // Outer
          else
            Label[e] := if isCritical[p] then "C" else "R";
          end;
        else
          Insert new vertex v with position p into GS
          Replace e by two edges e0 := (v, v0) and e1 := (v, v1);
          Label[v] := if isCritical[p] then "C" else "B";
          if Label[v0] = "O" then
            Label[v1] := "I";
            Label[e1] := if isCritical[p] then "C" else "R";
            Label[e0] := "O";
```

```
          else
            Label[v1] := Label[e1] := "O";
            Label[e0] := if isCritical[p] then "C" else "R";
          end;
        end;
      else // bisector is a parabolic arc
        Label[v1] := Label[v0];
        if Label[v0] = "O" then
          Label[e] = "O";
        else
          Label[e] = "I";
        end;
      end;
    end;
  else // Critical or Boundary
    if v1 is located to the right of c1 or c2 then
      Label[v1] := "Inner";
      Label[e] := if Label[v0] = "C" then "C" else "R";
    else
      Label[v1] := "Outer";
      Label[e] := if Label[v0] = "C" then "C" else "O";
    end;
  end;
```

3.3 COMPLEXITY ANALYSIS

The total complexity of the skeletonizing algorithm above is described in Theorem 1, which uses Lemmas 1-3 to establish the complexities of each individual step of the algorithm.

Lemma 1. The complexity of Step 1 of the skeletonizing algorithm is $O(N \log N)$, where N is a number of points in a polygon.

Proof. The Step 1 is about the construction of the Voronoi diagram for a line segments, which are extracted from the input polygons. The complexity of the Fortune's algorithm for Voronoi diagram construction algorithm according to [27] is $O(M \log M)$, where M is a number of line segments. Since N is proportional to M, the complexity of the Step 1 is $O(N \log N)$. ■

Lemma 2. The complexity of Step 2 of the skeletonizing algorithm is $O(N)$, where N is a number of the points in an input polygon.

Proof. Step 2 is about labeling the edges and vertices of the Voronoi graph using BFS traverse algorithm. Note that the Voronoi graph is a planar connected graph. Therefore, Euler's formula $|V| - |E| + f = 2$ takes place, where $|V|$, $|E|$, f is a number of vertices, edges and faces of a graph. If $|V| = N$, then the number of edges $|E| = O(N)$.

The BFS algorithm traverses all edges of the Voronoi graph. Since all operations within one BFS iteration can be performed in $O(1)$, the complexity of BFS routine is $O(|E| + |V|) = O(N)$.

Thus, the complexity of Step 2 is $O(N)$. ■

Lemma 3. The complexity of Steps 3-4 of the skeletonizing algorithm is $O(N)$, where N is a number of the points in an input polygon.

Proof. One edge can be removed from DCEL in $O(1)$ by reassigning the pointers [25, 28]. According to Lemma 2, the number of edges $|E| = O(N)$. Therefore, the complexity of Step 3 is $O(N)$. A single isolated vertex can be removed from DCEL in $O(1)$. Therefore, the complexity of Step 4 is $O(N)$. ■

Theorem 1. The complexity of the skeletonizing algorithm is $O(N \log N)$, where N is a number of the points in an input polygon.

Proof. According to analysis of the complexities of each algorithm's step provided in Lemmas 1-3, the total complexity of skeletonizing algorithm is $O(N \log N)$. ■

4. OPTIMIZATION

The purpose of this section is to introduce an optimization heuristic algorithm, which allow us to compute fast the Voronoi skeleton by reducing the number of vertices of input polygons. The main idea behind the optimized Voronoi skeleton construction algorithm is illustrated by the following lemma.

Lemma 4. Let $\mathcal{P} = \{p_1, p_2, \dots, p_N\}$ be a polygon and l_i denotes the line segment between points p_i and p_{i+1}, $i = 1, \dots, N$, $p_{N+1} = p_0$ of a polygon $\mathcal{P}$. The polygon $\mathcal{P}'$ is obtained by subdividing line segments $l_i, i = 1, \dots N$ of a polygon $\mathcal{P}$ such that the line segment l_i is replaced by a polyline formed by points $p_{i,1}, p_{i,2} \dots, p_{i,R_i}$ sampled on l_i, $i = 1,2, \dots, N$ ($p_{i,1} = p_i$, $p_{i,R_i} = p_{i+1}$). Then the Voronoi skeletons $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}')$ constructed using the skeletonizing algorithm above are equal (in terms of the Hausdorff distance between the corresponding Voronoi graphs).

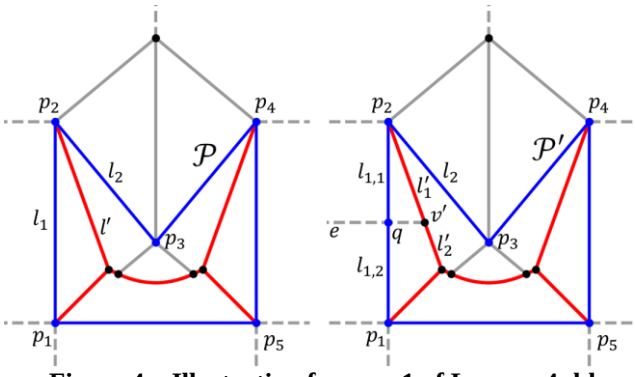

Figure 4 – Illustration for case 1 of Lemma 4: blue lines correspond to an input polygon, red edges are final Voronoi skeleton, gray and red edges compose Voronoi diagram for line segments

Proof. The Voronoi diagram of line segments of $\mathcal{P}$ and $\mathcal{P}'$ consists of the bisectors of the following types: a bisector between two line segment interiors, a bisector between a line segment interior and an endpoint and a bisector between two endpoints. Let us consider these cases separately:

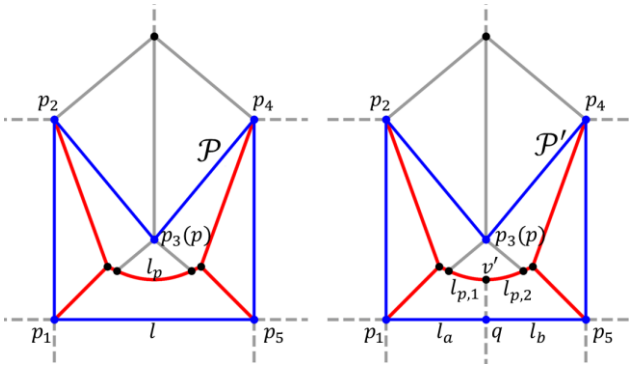

Figure 5 – Illustration for case 2 of Lemma 4: blue lines correspond to an input polygon, red edges are final Voronoi skeleton, gray and red edges compose Voronoi diagram for line segments

Case 1 (see Fig. 4). The bisector between two line segment interiors l_1 and l_2 is a line segment l' [27, 28]. Let us suppose that in $\mathcal{P}'$ line segment l_2 remains the same and l_1 is subdivided into two parts $l_{1,1}$ and $l_{1,2}$ connected by a shared endpoint q. Then, the Voronoi cell corresponding to l_1 in $\mathcal{VD}(\mathcal{P})$ will be split into two Voronoi cells (correspondingly $l_{1,1}$ and $l_{1,2}$) of $\mathcal{VD}(\mathcal{P}')$ by the Voronoi edge e such that e is a bisector between $l_{1,1}$ and $l_{1,2}$ which passes through q and is perpendicular to l_1 (and therefore, $l_{1,1}$ and $l_{1,2}$). Thus, the Voronoi edge e will divide bisector line segment l' in $\mathcal{VD}(\mathcal{P})$ into two parts l'_1 and l'_2 in $\mathcal{VD}(\mathcal{P}')$ such that l'_1 is a Voronoi edge of the Voronoi cell of $l_{1,1}$ and l'_2 is the Voronoi edge of the Voronoi cell of $l_{1,2}$. Note that l'_1, l'_2 and edge e are connected together by a newly introduced Voronoi vertex v'. The remaining part of the Voronoi diagrams for $\mathcal{P}'$ and $\mathcal{P}$ stays the same.

The BFS labeling procedure (see Step 2 of the algorithm above) for Voronoi edges and vertices of $\mathcal{VD}(\mathcal{P}')$ will split the introduced in $\mathcal{VD}(\mathcal{P}')$ Voronoi edge e into two parts e_1 and e_2: one part will be labeled as "Outer" and the other part will be labeled as "Redundant". Therefore, both parts will be removed at Step 3 of the skeletonizing algorithm and the resulting Voronoi skeleton $\mathcal{VS}(\mathcal{P}')$ will contain the line segment edges l'_1, l'_2 connected by v'.

Case 2. In case of a line segment interior l and an endpoint p, two possible scenarios take place. First scenario is when p is an endpoint of l. In this case Voronoi diagram contains an edge e' coming through p and perpendicular l. The edge e' can be either removed or not by BFS procedure depending on the type of p. Subdividing l into two parts l_a and l_b which share an endpoint q will introduce a new edge e parallel to e', which will be classified as "Redundant" and removed from the final skeleton. The second scenario (see Fig. 5) is when p is not an endpoint of l. Then the bisector between p and l is a parabolic arc l_p, which is subdivided into two parts $l_{p,1}$, $l_{p,2}$ if we split l into l_a and l_b. The analysis in this case is the similar to the Case 1 except that now l'_1 and l'_2 are parabolic arcs $l_{p,1}$ and $l_{p,2}$, respectively.

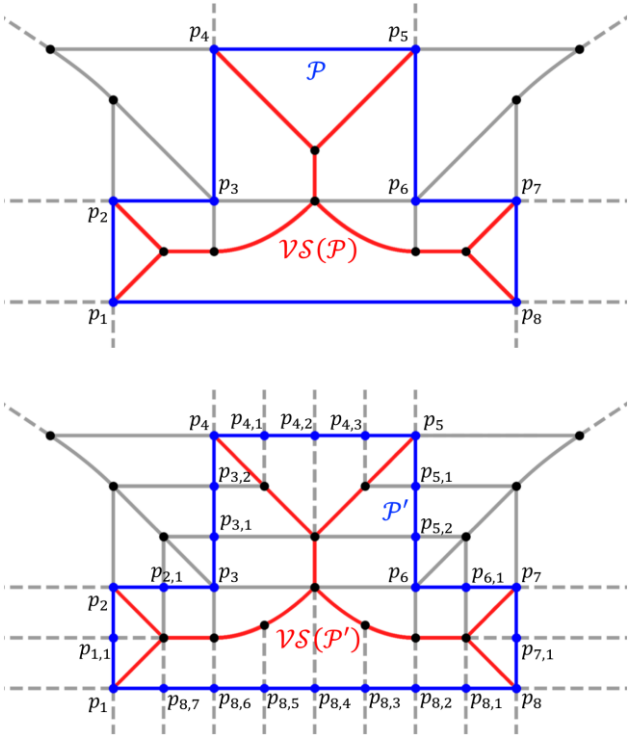

Figure 6 – The Voronoi skeletons (red) for polygon $\mathcal{P}$ (blue) and its subdivided version $\mathcal{P}'$ (blue) and respective Voronoi diagrams (gray)

Case 3. The bisector between two different endpoints of $\mathcal{VD}(\mathcal{P}')$ or $\mathcal{VD}(\mathcal{P})$ is an infinite edge (ray), which is classified at Step 2 of the algorithm above as "Outer" and, therefore, removed from both $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}')$ at Step 3.

The case of single subdivision ($L = 1$) of the polygon line segment for different possible bisectors of the Voronoi diagram is covered above. The general case for several subdivisions L can be proved by induction on L as described below.

Let us assume that for $L = n$ subdivisions of $\mathcal{P}$ hold that $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}')$ are equal. The polygon $\mathcal{P}''$ is obtained from $\mathcal{P}'$ by subdividing an arbitrary line segment of $\mathcal{P}'$ into two line segments. Therefore, we can apply one of the proved cases for a single subdivision above and obtain that Voronoi skeletons $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}'')$ are equal. Thus, by induction $\mathcal{VS}(\mathcal{P})$ and $\mathcal{VS}(\mathcal{P}')$ are equal for any $L > 0$. ■

Table 1. The overview of polygon (polyline) simplification algorithms

Name of algorithm	Avg. complexity	Worst-case complexity	Simpl. Req.
Ramer-Douglas-Peucker [30]	$O(N \log N)$	$O(N^2)$	yes
Visvalingam-Whyatt [31]	$O(N \log N)$	$O(N \log N)$	yes
Reumann-Witkam [32]	$O(N)$	$O(N)$	yes
Opheim [33]	$O(N)$	$O(N)$	yes
Lang [34]	$O(NK)$	$O(NK^2)$	yes
Zhao-Saalfeld [35]	O(N)	$O(N)$	yes
Rapso [36]	$O(N)$	$O(N)$	no
Li-Openshaw [37]	O(N)	O(N)	no
Nth point [38]	$O(N)$	$O(N)$	no
Circle [38]	$O(N)$	$O(N)$	no
Perpendicular distance [38]	$O(NK)$	$O(N)$	yes

Remark. It follows from Lemma 4 that the Voronoi skeleton $\mathcal{VS}(\mathcal{P}')$ for a subdivided polygon $\mathcal{P}'$ is the same (w.r.t. Hausdorff distance) as the Voronoi skeleton $\mathcal{VS}(\mathcal{P})$ for the original polygon $\mathcal{P}$ (see Fig. 6). However, in comparison to $\mathcal{VS}(\mathcal{P})$, $\mathcal{VS}(\mathcal{P}')$ is represented with a larger number of Voronoi edges and vertices. Therefore, the concept of the Voronoi skeleton with a minimal number of vertices and edges take place. By applying Lemma 4 in the reverse direction, one aims to reduce the number of vertices and edges in the Voronoi skeleton. This in turn allows us to reduce the execution time of Voronoi skeletonization algorithm and also to compress the resulting graph representation of a skeleton preserving its geometrical properties.

Table 2. Suitable polygon simplification algorithms, their parameter and heuristics

Algorithm name	Abbr.	Parameter(s)	Heuristics for 2nd parameter
Ramer-Douglas-Peucker	DP	$\varepsilon > 0$ – tolerance parameter;	No
Visvalingam-Whyatt	VW	$A > 0$ – minimum effective triangle area;	No
Reumann-Witkam	RW	$\varepsilon > 0$ – perpendicular distance tolerance;	No
Opheim	OP	$\varepsilon_{min}, \varepsilon_{max} > 0$ – distance tolerances;	$\varepsilon_{max} = +\infty$ (large number)
Lang	LA	$\varepsilon > 0$ – perpendicular distance tolerance; $R \in \mathbb{N}$ – fixed size search region;	$R = \theta \cdot N$, N – number of points; $\theta \in \{0.05, 0.1, 0.2, 0.25, 0.5, 1.0\}$.
Zhao-Saalfeld	ZS	$\varepsilon > 0$ – sector bound error;	No
Perpendicular distance	PD	$\varepsilon > 0$ – perpendicular distance tolerance; $K \in \mathbb{N}$ – number of repetitions;	$R = \theta \cdot N$, N – number of points; $\theta \in \{0.05, 0.1, 0.2, 0.25, 0.5, 1.0\}$.

Therefore, the operation reverse to subdivision – simplification, should be applied to polygon $\mathcal{P}'$ in order to obtain $\mathcal{P}$. According to Lemma 4 simplification procedure (algorithm) should meet the following requirement:

Simplification requirement: The polygon simplification heuristic should reduce the points corresponding to colinear consecutive line segments of the polygon. The polylines formed by such points should be replaced by a single line segment.

Thus, we introduce the Step 0 in the skeletonizing algorithm: simplify each polygon of a set $\mathcal{S}$ by reducing the points associated with colinear consecutive line segments of the corresponding polygon. This operation can be performed using one of the existing polygon simplification algorithms, which satisfies the simplification requirement above.

Analysis of simplification algorithms. We have analyzed the most commonly used algorithms for polygon (polyline) simplification and summarized the results in Table 1.

The aim of the mentioned in Table 1 simplification algorithms is to reduce the number of points representing the polygon (polyline). However, certain simplification strategies do not agree with the simplification requirement derived from Lemma 4. For example, a naive Nth point simplification [38] method merely removes each Nth point from a polygon ignoring its geometry. Another algorithm – Circle simplification [38], aims to group together points forming spatial clusters based on the distance

threshold and replace these clusters by a single representative. Li-Openshaw [37] and Rapso [36] algorithms simplify polyline based on spatial pixel (or hexagon-based) grid. The latter two algorithms rather solve the problem of polyline digitization (useful, for example, for solving the problem of optimal map rescaling). Therefore, we considered only the algorithms fulfilling the simplification requirement above (see Table 2). Note that most algorithms in Table 1 have linear complexity (except Ramer-Douglas-Peucker [30] and Visvalingam-Whyatt [31], which have $O(N \log N)$ complexity). The open issue is to choose the simplification algorithm, which would allow us to achieve the best performance improvement showing the minimum influence on the resulting skeleton. This issue is empirically investigated in the following evaluation section.

5. ALGORITHM EVALUATION

In this section we evaluate the performance of skeletonization algorithm in terms of execution time and measure the influence of the heuristic optimization step onto the accuracy and execution time of the overall algorithm. We also evaluate the computational overheads related to suitable line simplification algorithms.

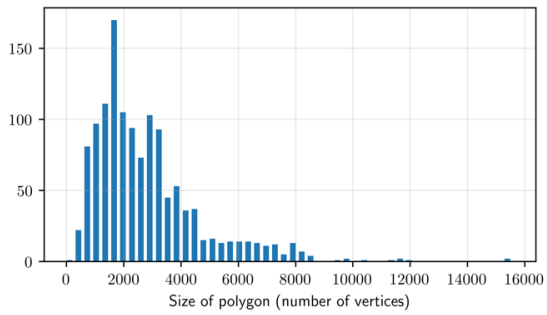

Figure 7 – Distribution of polygon's sizes (number of vertices)

Dataset. In order to empirically evaluate the performance of the overall skeletonization algorithm and individual heuristic simplification algorithms we used polygons obtained from the dataset MPEG 7 CE-Shape-1. These polygons were extracted from binary images using Marching Squares algorithm [19]. In total our dataset contains 1282 polygons. The distribution of polygon sizes (number of vertices) is shown in Fig. 7.

Measures. In the performed experiments we have measured the following quantities:

1. *Execution time* (ms) of each simplification algorithm, skeletonizing algorithm with (without) the mentioned heuristics and the total execution time. The experiments were carried out on Intel Core i7, 2.2GHz, 16Gb RAM.
2. *Hausdorff distances* d_H (errors) [39] between the simplified polygon and original polygon and also between the ground truth skeleton and the skeleton obtained using the skeletonization with a simplification heuristic;
3. *Simplification rate* (%) of the polygon is computed as follows:

$$SR(P, P') = \frac{|P|-|P'|}{|P|} \cdot 100\%, \qquad (3)$$

where P is an original polygon, P' is a simplified polygon and $|P|$ is a number of vertices of a polygon P. High simplification rate means that simplified polygon has a small number of vertices in comparison to the original one.

Parameters. The parameters of the simplification algorithms (see Table 2) were chosen using the line search method such that the maximum simplification rate is achieved for a given threshold value of the Hausdorff error d_H – the distance between simplified polygon P' and an original polygon P. This allows us to compare different simplification algorithms with respect to the maximum allowed error. The established parameters of the simplification algorithms for the respective values of d_H are shown in Table 3.

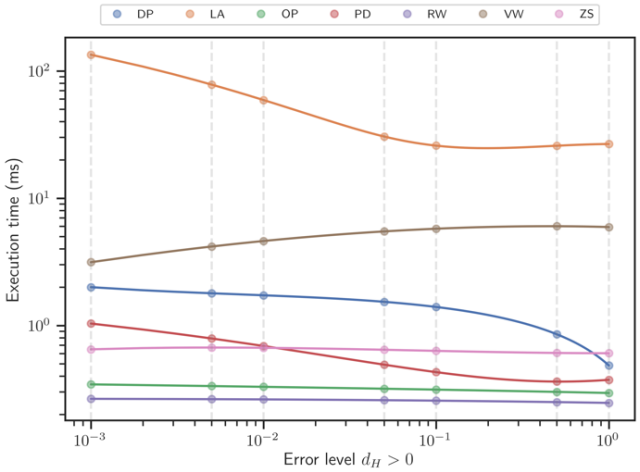

Figure 8 – Execution time of the simplification algorithms

For the algorithms with two parameters we applied the heuristics to choose the value of the second parameter as described in Table 2. These

heuristics were designed to achieve the maximum simplification rate for a given Hausdorff error threshold d_H. It was established that for the algorithms of Lang and "Perpendicular distance" the optimal value of θ is 0.25 and for $\theta > 0.25$ the simplification rate does not increase (however, the execution time of these simplification algorithms increases leading to additional overhead).

Evaluation results. Using the polygons from the dataset MPEG 7 CE-Shape-1 we have measured the execution time of each suitable simplification algorithm in relation to the Hausdorff error threshold d_H (see Fig. 8). These measurements can be considered as a computational overhead related to the optimization step of our skeletonizing algorithm.

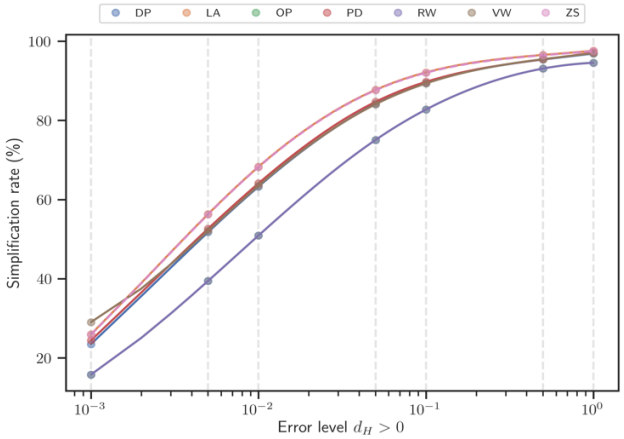

Figure 9 – Simplification rates depending on the error threshold. Nearly identical curves are: (LA, ZS); (PD, VW, DP); (OP, RW)

In order to compare the quality of the simplification algorithms, we have measured the dependence of the simplification rate on the error threshold value d_H.

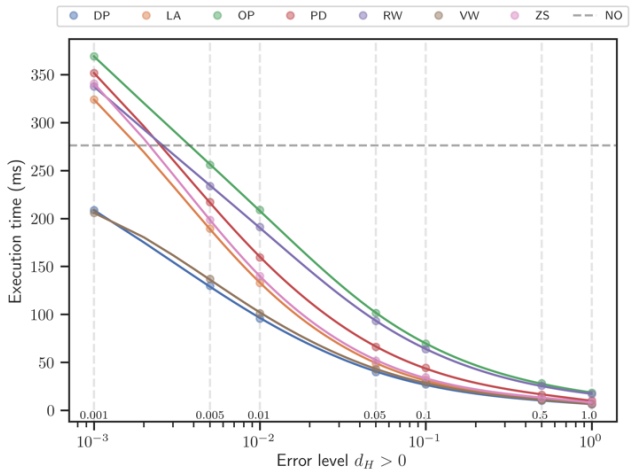

Figure 10 – Execution time of skeletonization routine with heuristics, horizontal dash line (NO) shows execution time of skeletonization without optimization

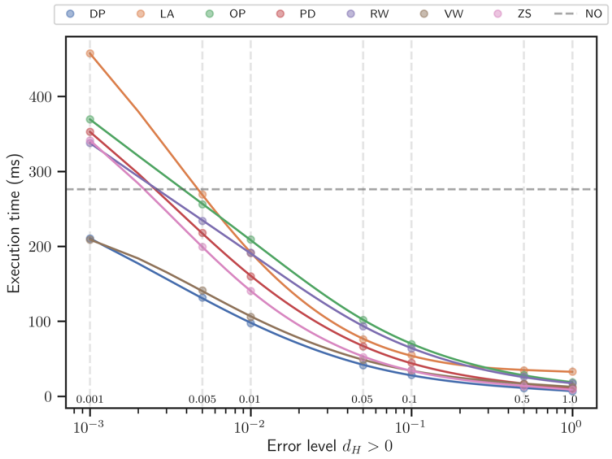

Figure 11 – Overall execution time. Horizontal dash line (NO) shows execution time of skeletonization without optimization

Fig. 9 shows that in general the largest polygon simplification for a given d_H is achieved by the algorithms LA and ZS, which have approximately identical dependency curves. Slightly smaller simplification is accomplished using the algorithms PD, VW and PD, which show also almost undistinguishable behavior (except for the algorithm VW, which overperforms all other algorithms for small values of $d_H < 0.002$). The lowest simplification rates are attained by OP and RW algorithms with nearly identical dependency curves.

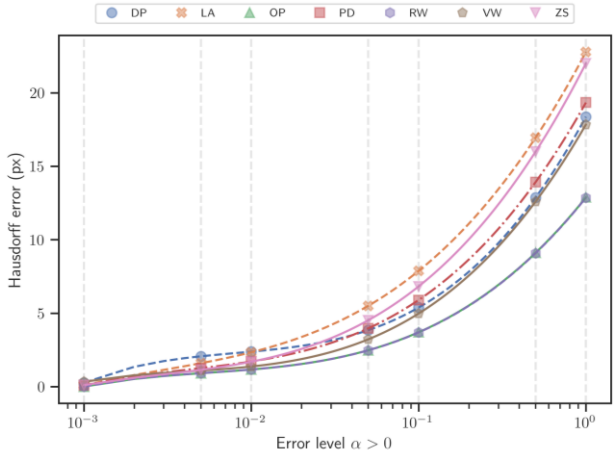

Figure 12 – The Hausdorff error of skeletonization algorithm in relation to the simplification error achieved by different simplification algorithms (curves for RW and OP are nearly identical)

As we can observe from Fig. 9 and Fig. 10, the fastest algorithms OP and RW achieve the smallest simplification rate and, therefore, cannot guarantee the fastest performance of the skeletonization algorithm. Thus, we have measured the execution time of the proposed skeletonization algorithm using the simplified polygons. The results are shown in Fig. 8.

In order to take into account the overhead execution time of the simplification algorithms, we

have measured the total execution time of the skeletonization algorithm including the respective simplification routines (see Fig. 11).

Fig. 11 allows us to choose the fastest version of the optimization heuristics. However, we need to consider that the heuristic optimization step might affect the accuracy of the resulting skeleton. Therefore, we have also calculated the error of the optimized skeletonization algorithm. Such error is measured as the Hausdorff distance between the ground truth skeleton and the result of the optimized skeletonization algorithm (see Fig. 10).

6. DISCUSSION

Fig. 11 shows that Ramer-Douglas-Peucker (DP) and Visvalingam-Whyatt (VW) algorithms allow us to speed-up our skeletonization method to the greatest extent. These two algorithms (only) overperform the optimization-free approach (NO) in case of small values of $d_H \leq 0.001$.

The skeletonization based on Opheim and Reumann-Witkam algorithms exposes the smallest error among the other approaches (see Fig. 12). However, for $d_H < 0.002$ these algorithms have a large computational overhead eliminating the effect of the optimization. Therefore, it is reasonable to apply them only for $d_H > 0.002$. Note that the difference between skeletonizing errors decreasing as d_H becomes smaller (see Fig. 12).

We have computed 2-sample t-test to validate the hypothesis that algorithms DP and VW produce different average skeletonization errors. The hypothesis testing (see Table 4) showed that the skeletonizing errors produced by DP and VW are undistinguishable.

Table 3. Parameters of the simplification algorithms

Hausdorff distance d_H	Algorithm parameters						
	DP	VW	RW	OP	LA (0.25)	ZS	PD (0.25)
0.001	0.001	0.0007	0.001	0.001	0.001	0.001	0.001
0.005	0.005	0.0025	0.005	0.005	0.005	0.005	0.005
0.01	0.01	0.005	0.009	0.009	0.01	0.01	0.01
0.05	0.05	0.025	0.04	0.04	0.05	0.05	0.05
0.1	0.1	0.05	0.08	0.08	0.1	0.1	0.1
0.5	0.5	0.25	0.4	0.4	0.5	0.5	0.5
1.0	1	0.5	0.8	0.8	1	1	1

Another hypothesis testing was performed to distinguish between the execution time between DP and VW algorithms. Table 5 and Fig. 11 show that for the most of the cases (except $d_H = 0.001$) DP algorithm executes faster than VW.

Speed-accuracy trade-off. Since none of the tested algorithms minimizes the accuracy and execution time of the skeletonizing method at the same time, the optimal heuristic choice should base on the trade-off between accuracy and the speed. Thus, based on the performed computational experiments the following conclusions can be drawn:

1. If accuracy of the resulting skeleton is critical, then for $d_H > 0.002$ the optimization can be performed using OP or RW algorithms. However, for $d_H < 0.002$ the only reasonable optimization is using the DP or VW algorithms;
2. If execution time of the algorithm is more critical than the accuracy, then optimization can be performed using DP or VW algorithms, which according to the provided experiments give 1.7 times less accurate result then RW and OP heuristics;

Pruning effect of polygon simplification. It was experimentally discovered, that the introduced optimization heuristics influences the skeleton in a similar way as pruning methods [40]. Fig. 13 shows that for large values of d_H simplification heuristics tends to regularize shape of the object in a way that the branches of the skeleton corresponding to small shape perturbation disappear. Therefore, such optimization allows us not only to speed-up the execution of the skeletonization, but also to achieve a pruning effect and remove the noisy brunches of skeleton.

Table 4. Testing the hypothesis that DP and VW produce different average Hausdorff errors (curved p-values are above the significance level 0.05, underlined – below).

t-Test	Level threshold $d_H > 0$						
	0.001	0.005	0.01	0.05	0.1	0.5	1.0
t-statistic / p-value	-1.09 / *0.2779*	8.40 / $\underline{1.13 \cdot 10^{-16}}$	8.01 / $\underline{2.43 \cdot 10^{-15}}$	1.90 / *0.0580*	1.04 / *0.2974*	0.59 / *0.5544*	0.56 / *0.5751*

Table 5. Testing the hypothesis that DP and VW have different average execution time (curved p-values are above the significance level 0.05, underlined – below).

t-Test	Level threshold $d_H > 0$						
	0.001	0.005	0.01	0.05	0.1	0.5	1.0
t-statistic / p-value	0.26 \ *0.7912*	-2.21 \ 0.0275	-2.48 \ 0.01323	-4.73 \ $2.34 \cdot 10^{-6}$	-6.01 \ $2.2 \cdot 10^{-9}$	-13.25 \ $8.46 \cdot 10^{-39}$	-19.34 \ $6.51 \cdot 10^{-78}$

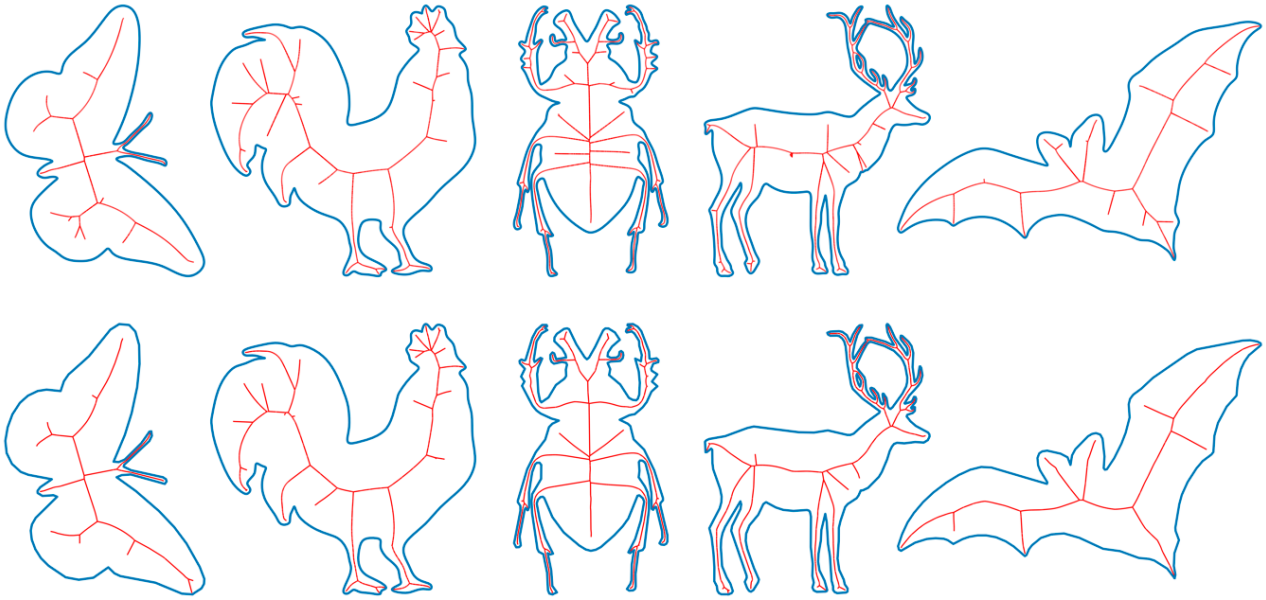

Figure 13 – Examples of optimized Voronoi skeletons for arbitrary shapes from MPEG 7 CE-Shape-1 dataset. Optimization heuristics is DP. $\mathbf{d_H = 0.001}$ for the top row of images, $\mathbf{d_H = 1.0}$ for the bottom row of images

6. CONCLUSION

In this paper, we proposed an optimized algorithm for computing the Voronoi skeleton based on polygonal data. This topic is of relevance because of its direct relation to the optimization tasks in image processing and computer graphics (in particular, the efficient processing of vectorized images). We have illustrated in detail the main steps of the proposed skeletonization algorithm. It was established that the complexity of the algorithm is $O(N \log N)$, where N is the number of vertices in a polygon. We have also proposed theoretically justified optimization heuristic, which is based on polygon/polyline simplification algorithms. In order to evaluate and prove the efficiency of such heuristic, a series of computational experiments were conducted based on the polygons obtained from MPEG 7 CE-Shape-1 dataset, which represent the most commonly observed shapes in computer graphics and vision. In order to determine the most suitable optimization heuristic, we have evaluated seven different appropriate state-of-the-art simplification algorithms. We have measured the execution time of the skeletonization algorithm with and without the optimization and determined the computational overheads related to such optimizations. Also, we determined the accuracy of the optimized skeletonization algorithm depending on the applied optimization. As a result, we have established the criteria, which allow us to choose the optimal heuristics depending on the system's requirements.

7. REFERENCES

[1] P. K. Saha, G. Borgefors, G. S. Baja, "A survey on skeletonization algorithms and their applications," *Pattern Recognition Letters*, vol. 76, pp. 3-12, 2016.

[2] H. Sundar, D. Silver, N. Gagvani, S. Dickinson, "Skeleton based shape matching and retrieval," *2003 Shape Modeling International*, Seoul, South Korea, 2003, pp. 130-139.

[3] J. Xie, P. Heng, M. Shah, "Shape matching and modeling using skeletal context," *Pattern Recognition*, vol. 41, issue 5, pp. 1756-1767, 2008. doi:10.1016/j.patcog.2007.11.005.

[4] A. Chaudhuri, K. Mandaviya, P. Badelia, S. Ghosh, *Optical Character Recognition*, Springer, Cham, 2017, 248 p.

[5] R. S. Torres, A.X. Falcão, "Contour salience descriptors for effective image retrieval and analysis," *Image and Vision Computing*, vol. 25, issue 1, pp. 3-13, 2007.

[6] K. Rezaee, J. Haddadnia, A. Tashk, "Optimized clinical segmentation of retinal blood vessels by using combination of adaptive filtering, fuzzy entropy and skeletonization," *Applied Soft Computing*, vol. 52, pp. 937-951, 2017. doi: 10.1016/j.asoc.2016.09.033.

[7] W. Lasso, Y. Morales and C. Torres, "Image segmentation blood vessel of retinal using conventional filters, Gabor transform and skeletonization," *Proceedings of the 19th Symposium on Image, Signal Processing and Artificial Vision*, Armenia, Colombia, September 17-19, 2014, pp. 1-4. doi: 10.1109/STSIVA.2014.7010170

[8] Y. Al-Kofahi, N. Dowell-Mesfin, C. Pace, W. Shain, J. N. Turner, B. Roysam, "Improved detection of branching points in algorithms for automated neuron tracing from 3D confocal images", *Cytometry*, vol. 73, pp. 36-43, 2008. doi: 10.1002/cyto.a.20499

[9] C. Faulkner, J. Zhou, A. Evrard, G. Bourdais, D. MacLean, H. Häweker, P. Eckes, S. Robatzek, "An automated quantitative image analysis tool for the identification of microtubule patterns in plants," *Traffic*, vol. 18, pp. 683–693, 2017. doi: 10.1111/tra.12505

[10] M. Beil, H. Braxmeier, F. Fleischer, V. Schmidt, P. Walther, "Quantitative analysis of keratin filament networks in scanning electron microscopy images of cancer cells," *Journal of Microscopy*, Vol. 220, pp. 84-95, 2005. doi: 10.1111/j.1365-2818.2005.01505.x

[11] S. Changxian, M. Yulong, "Morphological thinning based on image's edges," *Proceedings of the 1998 International Conference on Communication Technology*, Beijing, China, October 22-24, 1998, pp. 5-10. doi: 10.1109/ICCT.1998.743232

[12] T. Y. Zhang, C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, issue 3, pp. 236-239, 1984.

[13] T. Q. Yan, C. X. Zhou, "A continuous skeletonization method based on distance transform," in: D. S. Huang, P. Gupta, X. Zhang, P. Premaratne (Eds.), *Emerging Intelligent Computing Technology and Applications. Communications in Computer and Information Science*, Springer, Berlin, 2012, pp. 251-258. doi: 10.1007/s00371-018-1549-z

[14] J. Chen, M. Du, X. Qin, "An improved topology extraction approach for vectorization of sketchy line drawings," *The Visual Computer*, vol. 34, issue 12, pp. 1633–1644, 2018.

[15] X. Hilaire, K. Tombre, "Robust and accurate vectorization of line drawings," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, issue 6, pp. 890-904, 2006. doi: 10.1109/TPAMI.2006.127

[16] L. Acciai, P. Soda, G. Iannello, "Automated neuron tracing methods: An updated account," *Neuroinformatics*, Vol. 14, Issue 4, pp. 353–367, 2016.

[17] L. Cheng, J. De, X. Zhang, F. Lin, H. Li, K. H. Ong, W. Yu, Y. Yu, S. Ahmed, "A graph-theoretical approach for tracing filamentary structures in neuronal and retinal images," *IEEE Transactions on Medical Imaging*, vol. 35, issue 1, pp. 257-272, 2016.

[18] A. M. Stein, D. A. Vader, L. M. Jawerth, D. A. Weitz, L. M. Sander, "An algorithm for extracting the network geometry of three-dimensional collagen gels," *Journal of Microscopy*, vol. 232, pp. 463-475, 2008.

[19] C. Maple, "Geometric design and space planning using the marching squares and marching cube algorithms," *Proceedings of the 2003 International Conference on Geometric Modeling and Graphics*, London, UK, July 16-18, 2003, pp. 90–95.

[20] O. Aichholzer, F. Aurenhammer, D. Alberts, B. Gärtner, "A novel type of skeleton for polygons," *Journal of Universal Computer Science*, vol. 1, issue 12, pp. 752-761, 1995.

[21] D. Eppstein, J. Erickson, "Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions," *Discrete & Computational Geometry*, vol. 22, issue 4, pp. 569-592, 1999.

[22] F. Chin, J. Snoeyink, C. A. Wang, "Finding the medial axis of a simple polygon in linear time," *Proceedings of the 6th Annual International Symposium on Algorithms and Computation*, Cairns, Australia, December 4-6, 1995, pp. 382-391.

[23] R. Ogniewicz, M. Ilg, "Voronoi skeletons: theory and applications," *Proceedings of the 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Champaign, USA, 1992, pp. 63-69. doi: 10.1109/CVPR.1992.223226

[24] G. Székely, "Voronoi skeletons," in: K. Siddiqi, S. M. Pizer (Eds.), *Medial Representations. Computational Imaging and Vision*, Springer, Dordrecht, 2008, pp. 191-221.

[25] F. P. Preparata, M. I. Shamos, *Computational Geometry: An introduction*, first ed., Springer, Berlin, Heidelberg, 1985, 390 p.

[26] M. I. Shamos, D. Hoey, "Closest-point problems," *Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer*

Science, Berkley, USA, October 13-15, 1975, pp. 151-162.

[27] S. Fortune, "A sweepline algorithm for Voronoi diagrams," *Algorithmica*, vol. 2, pp.153-174, 1987. doi:10.1007/BF01840357

[28] M. Berg, O. Cheong, M. Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications*, third ed., Springer, Berlin, 2008, 386 p. doi:10.1007/978-3-540-77974-2

[29] A. Okabe, B. Boots, K. Sugihara, *Spatial Tessellations, Concepts and Applications of Voronoi diagrams*, second ed., John Wiley & Sons, New York, 2000, 696 p.

[30] D. Douglas, T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *The Canadian Cartographer*, vol. 10, issue 2, pp. 112–122, 1973.

[31] M. Visvalingam, J. D. Whyatt, "Line generalisation by repeated elimination of points," *Cartographic Journal*, Vol. 30, pp. 46-51, 1993.

[32] K. Reumann, A. P. M. Witkam, *Optimizing curve segmentation in computer graphics*, in A. Gunther, B. Levrat, H. Lipps (Eds.), *International Computing Symposium*, American Elsevier, North Holland, 1973, pp. 467–472.

[33] H. Opheim. "Fast data reduction of a digitized curve," *Geo-Processing*, vol. 2, pp. 33-40, 1982.

[34] T. Lang, "Rules for robot draughtsman," *Geographical Magazine*, vol. 42, pp. 50-51, 1969.

[35] Z. Zhao, A. Saalfeld, "Linear-time sleeve-fitting polyline simplification algorithms," *Proceedings of the Annual Convention and Exposition Technical Papers*, Seattle, USA, April 7-10, 1997, pp. 214-223.

[36] P. Raposo, "Scale-specific automated line simplification by vertex clustering on a hexagonal tessellation," *Cartography and Geographic Information Science*, vol. 40, issue 5, pp. 427-443, 2013.

[37] Z. Li, S. Openshaw. "Linear Feature's Self-Adapted Generalization Algorithm Based on Impersonality Generalized Natural Law," *Translation of Wuhan Technical University of Surveying and Mapping*, vol. 1, pp. 49-58, 1994.

[38] J. Song, R. Miao, "A novel evaluation approach for line simplification algorithms towards vector map visualization," *International Journal of Geo-Information*, vol. 5, issue 12, pp. 223-236, 2016.

[39] A. A. Taha, A. Hanbury, "An efficient algorithm for calculating the exact Hausdorff distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, issue 11, pp. 2153-2163, 2015.

[40] A. Beristain, M. Graña, A. I. Gonzalez, "A pruning algorithm for stable voronoi skeletons," *Journal of Mathematical Imaging and Vision*, vol. 42, pp. 225-237, 2012.

M.Sc., Dmytro Kotsur, *Post graduate student at the department of Mathematical Informatics, Faculty of Computer Science and Cybernetics, Taras Shevchenko National University of Kyiv. Graduated in 2014 at Faculty of Computer Science and Cybernetics at Taras Shevchenko National University of Kyiv, specialty – informatics. Areas of scientific interests: computational geometry, computer graphics, computer vision, pattern recognition, theory of algorithms, medical image processing, machine learning.*

Prof. Vasyl Tereshchenko, *Head of the department of Mathematical Informatics at Faculty of Computer Science and Cybernetics of Taras Shevchenko National University of Kyiv. Graduated in 1986 Mathematics and Mechanics Faculty at Taras Shevchenko National University of Kyiv, specialty – applied mathematics and mechanics. Areas of scientific interests: simulation and visualization, computational geometry, computer graphics, computer vision, pattern recognition, theory of algorithms, parallel algorithms and programming, information systems, database, nonlinear integral and differential equations, thermo mechanics inhomogeneous solids.*