Image processing to detect worms

Javier Fernández

June 27, 2010

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.2 Thesis motivation and purpose

1.3 The problem

1.4 Objectives

1.4.1 General Objectives

1.4.2 Specific Objectives

Chapter 2

Theoretical Framework

2.1 Endrov

2.2 Thresholding

Thresholding is a process of image segmentation that can be used to create binary images from gray-scale images. A binary image is a type of discrete image in which the value of a pixel is either 1 or 0 depending on whether the pixel belongs to the foreground or to the background.

As stated on [24], during the thresholding process individual pixels in an image are marked as “object” pixels if their value is greater than some threshold value (assuming an object to be brighter than the background) and as “background” pixels otherwise. This convention is known as *threshold above*. Variants include *threshold below*, which is opposite of threshold above; *threshold inside*, where a pixel is labeled ”object” if its value is between two thresholds; and *threshold outside*, which is the opposite of *threshold inside* [14]. Typically, an object pixel is given a value of 0 while a background pixel is given a value of 1 Finally, a binary image is created by coloring each pixel white or black, depending on a pixel’s label.

In image processing applications where the study is focused on particular objects contained in an image, thresholding becomes an effective and simple tool to separate these objects from the background. Commonly, the gray levels belonging to the object are substantially different from the gray levels of the background pixels. In [13, p.146] many thresholding applications on image processing are mentioned such as: document image analysis, where the goal is to extract printed characters, logos, graphical content, or musical scores; map processing where lines, legends and characters are to be found; scene processing, where a target is to be detected; and quality inspection of materials, where defective parts must be delineated, among many others.

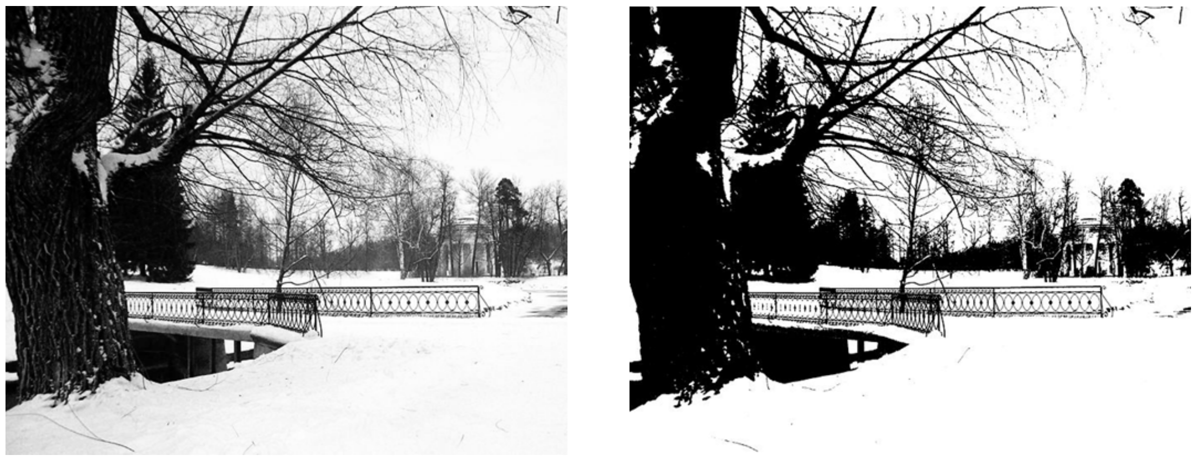
The key parameter in the thresholding process is the thresholding value (or values for *threshold inside approach*). The value can be automatically computed, what is called *automatic thresholding*, as well as set or tuned through user input.
According to the information they are exploiting, the different thresholding methods can be categorized. In [13, p.147], Sezgin and Sankur categorize the thresholding methods in six groups:

- Histogram shape-based methods: the peaks, valleys and curvatures of the smoothed histogram are analyzed
- Clustering-based methods: gray-level samples are clustered in two parts as background and foreground, or modeled as a mixture of two Gaussians.
- Entropy-based methods: algorithms that use the entropy of the foreground and background regions, the cross-entropy between the original and the binary image, etc.
- Object attribute-based methods:search a measure of similarity between the gray-level and the binarized images, such as fuzzy shape similarity, edge coincidence, etc.
- Spatial methods: use higher-order probability distribution and/or correlation between pixels
- Local methods: adapt the threshold value on each pixel to the local image characteristics.

Below, in fig.2.1, two images are shown that correspond to a gray-scale image and binary image obtained by thresholding.

2.3 Distance Transform

A distance transform or distance map is a representation of a digital image that is obtained by converting a digital binary image, consisting in object and non-object pixels, to another



(a) Gray-scale Image (b) Binary image obtained through thresholding

Figure 2.1: Gray-scale Image before and after a thresholding effect is applied. Images taken from [24]

image in which each pixel has a value corresponding to the distance to the nearest non-object pixel. The object pixels can be considered as foreground and the non-object pixels as background. The obtained image is then a sort of gray-scale representation of the foreground pixels in the binary image.
The pixel mapping depends mainly on the distance metric, which is the measurement method of distance between image pixels. Different metrics have been studied to find distance maps such as *City Block or Manhattan*, *Chessboard*, *Euclidean*, *Chamfer 3-4*, *Octagonal*, among others.[4, p.363]. There exist a great amount of distance metric of other kinds, that are useful for different purposes. These are commonly derived from the previous. Below, Fig.2.2 shows the images obtained by applying different distance metrics.

As stated on [6] Distance transforms play a central role in the comparison of binary images, particularly for images resulting from local feature detection techniques such as edge or corner detection. For example, both the Chamfer and Hausdorff matching approaches make use of distance transforms in comparing binary images. Distance maps can also be interpreted as landscapes of islands where the label of every pixel indicates the height of the region. This allows the detection of ridges and peaks which is a straightforward way to find the skeleton of an object.[1, 237]. The nature of distance transforms in which the objects are represented as contour layers of different depth makes them also a useful tool for edge analysis and to improve efficiency of morphology algorithms such as *Thinning* and

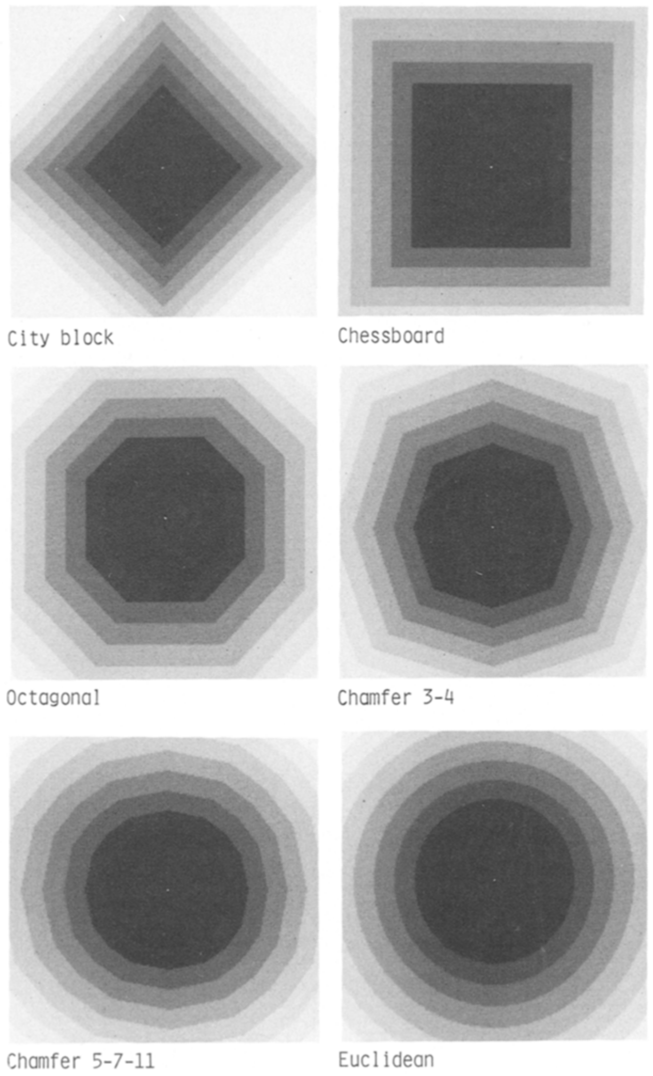
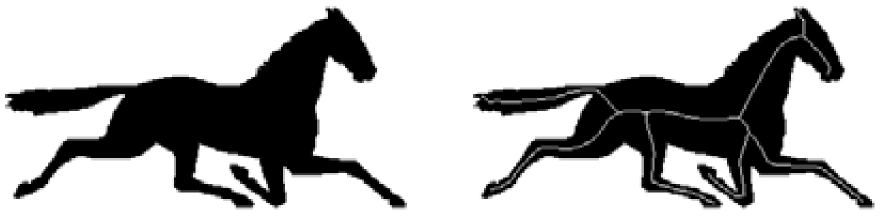


Figure 2.2: The distances from a point for the six DTs. The lighter the color the larger the distance [4, p.365]

Thickening.

2.4 Skeletonization

A skeleton is a compact and simple representation of an object that consists in a thin version of it that is equidistant to its boundaries and preserves many of the topological and geometrical characteristics of the original image, as explained in [23, 8, 17]. Usually the skeleton is defined as the centers of maximal discs contained in the original image, [8, 17]. Regardless of the definition that is adopted, if the skeleton points are attributed with their distances to the original boundary of the object, the skeleton can be used to exactly reconstruct the original shape. Figure 2.3 shows a skeleton calculated from a horse shape and the original binary image.



(a) Binary Image (b) Skeleton obtained using the ssm of the distance transform[8]

Figure 2.3: Horse binary shape and skeleton obtained using the ssm of the distance transform. Images taken from [8]

Depending on the way they are produced, skeletons can be categorized in different types. Telea et al describe three types in [17], such as: *Morphological Thinning*, *Geometric Methods* and *Distance transform.* The *Morphological Thinning* methods itereratively peel off (or reduce) the boundary layer by layer, identifying points whose removal does not affect the object's topology. These are usually straightforward and usually require intricate heuristics to ensure the skeletal connectivity, as mentioned in [17]. Two fast parallel approaches that ensure connectivity using the *Morphological thinning* method are described in [5] and [26]. The *Geometric Methods* compute the Voronoi diagram of a discrete polyline-like sampling of the boundary. A Voronoi diagram is the boundary's medial axis. "Such methods produce an accurate connected skeleton, but are fairly complex to implement, require a robust boundary discretization, and are computationally expensive" [17, p.251]. The third defined method computes *the distance transform* (see Sec. 2.3) of the objects boundary. The common approach consists in finding the ridge points and connecting them [16, 2, 3]. Usually they can ensure the accurate localization of skeleton points but neither connectivity or completeness.

Skeletons are important for object representation and recognition in different areas, such as: computer vision, image analysis, and digital image processing, including optical character recognition, fingerprint recognition, visual inspection, pattern recognition, binary image compression, and protein folding, as mentioned on [12].

2.5 Image Segmentation

2.6 Shape Matching

Shape matching is a central problem in visual information systems, computer vision, pattern recognition, and robotics [21]. It consists in identifying the area or contour of a specific shape or class of shapes in an image, and plays a fundamental role in content extraction from images and content-based image retrieval. In [20] Veltkamp explains that shape matching deals with transforming a shape and measuring the resemblance with another one, using some similarity measure, that normally correspond to the notion of distance between shapes.
The concept of shape is abstract, but most approaches in shape matching represent a shape as a geometrical object. This can be both a set of points, curves, surfaces, solids,etc. and a geometrical pattern modulo some transformation group, in particular similarity transformations (translations, rotations and scalings), as it is stated on [20]. Usually a geometrical pattern of a shape called shape descriptor is used to represent the class of the matching object. There are different types of shape descriptors depending on the information they supply and the nature of the problem (see Sec.2.7)

There are different studied ways to approach the shape matching problem. Since the approach followed in this thesis work is related to those that deal with computational geometry, the emphasis on this section will be on the last ones. Computational Geometry studies algorithms that can be declared in terms of geometry.

In [21] Veltkamp and Hagedoorn mention different approaches of shape matching such as: tree pruning, the generalized Hough transform or pose clustering, geometric hashing, the alignment method, statistics, deformable templates, relaxation labeling, Fourier descriptors, wavelet transform, curvature scale space and neural networks. They also categorize the matching techniques in two main groups: *global image transforms* and *global objects methods*. The *global image transform* group refers to the techniques that "transform the image from color information in the spatial domain to color variation in the frequency domain". These approaches do not represent the shape explicitly for matching, instead they represent color or intensity transitions in the image. This makes impossible to measure the difference of two images in terms of shape as well as match a shape with a specific part of an image. On the other hand the *global object methods* work with a complete object area or contour and can analyze specific areas in the image instead of requiring to process the whole image as in the global image transforms. In order to perform a proper matching, the objects in the image have to be completely and clearly segmented. Some of these methods are: *moments*, where an object is described as a set of moments, *modal matching*, where the boundary is used instead of the area and is described with Fourier descriptors and *curvature scale space*, where a scale space and parameterized representation of the contour of the objects is used.

Veltkamp describes in [20] four different forms in which shape matching is studied, given two shape patterns and a dissimilarity measure. These are:

- **Computation Problem:** Compute the dissimilarity between the two patterns
- **Decision Problem:** For a given threshold, decide whether the dissimilarity is smaller than the threshold
- **Decision Problem:** For a given threshold, decide whether there exists a transformation such that the dissimilarity between the transformed pattern and the other pattern is smaller than the threshold
- **Optimization Problem:** Find the transformation that minimizes the dissimilarity between the transformed pattern and the other pattern.

A well studied optimization approach for shape matching is Active Contour Models (*Snakes*), in which is inspired much of the shape fitting approach of this work (see Sec 3.2.9).In [9]

a snake is defined as an energy-minimizing spline guided by external constraint forces and influenced by image forces that pull it toward features such as lines and edges. The *snakes* are said to be active contour models because they lock onto nearby edges, localizing them accurately.
The *snakes* model is defined as a controlled continuity spline that is bound to internal and external image forces, called energies. The external energy models how well the deformed model matches the data. The internal energy models the objects resistance to be pushed by the external force into directions not coherent with the prior knowledge, as it is stated on [19].The internal energy imposes a "piecewise smoothness constraint",[9]. This means that a contour is pushed to an image feature by the external force while the contour itself exhibits resistance to be deformed into a non-smooth curve.

Given these definitions, let M be the model and D a data set the energy E can be defined as:

$$E(M) = E_{ext}(M, D) + E_{int}(M)$$

where E_{ext} is the external energy function and E_{int} the internal energy function.As it is explained on [19] the image forces push the snake toward salient image features like line, edges and subjective contours, while the external constraint forces are responsible for putting the snake near the desired local minimum. The optimization algorithm consists then in minimizing the objective function until the best solution is found.

2.7 Shape descriptor

A shape descriptor is a structured abstraction of a class of shapes that describes them in geometrical terms. Shape descriptors can be constructed as either fixed or variable geometrical shape. If the descriptor is of the variable type then depending on the different values assigned to its parameters, different shapes are generated but that still belong to the same type or class of shapes. As stated on [18] shape models have been used widely to achieve robust interpretation of complex images. They allow image evidence to be organized into plausible interpretations which can then be verified.

In [7] Latecki et al divide shape descriptors into three main categories:

- **Contour based descriptors:** The contour of a given object is mapped to some representation from which a shape descriptor is derived
- **Image based descriptors:** The computation of a shape descriptor is based on summing up pixel vales in a digital image containing the silhouette of a given object; the shape descriptor is a vector of a certain number of parameters derived this way
- **Skeleton based descriptors:** After a skeleton is computed, it is mapped to a tree structure that forms the shape descriptor; the shape similarity is computed by some tree-matching algorithm

Considering that basically shape descriptors are "attempts to quantify shape in ways that agree with human intuition"[10, p.1] any kind of geometrical interpretation that covers the contents or properties that want to be described on a shape, can be used as a shape descriptor. In [10] region-based shape descriptors are covered which are such that attempt to describe a shape based on the geometrical and numerical properties of the region of the shape. First some simple descriptors are mentioned such as: area, perimeter, (Non-)Compactness or (Non-)Circularity, Eccentricity, Elongation, Rectangularity and Orientation. Any combination of this properties of a shape are useful to describe them in a basic and general way. Other more complex properties are mentioned to improve the accuracy of the descriptor: *convex hull*, *extremal points*, *profiles*, *moments* and *profile moments*. *Convex hull or bays* describes the shape by measuring the number or size of concavities in the shape. *Extremal points* is based on finding the points that are at the extreme of the shape. This can be a simple representation as the *bounding box* or a more powerful as it is finding the eight extremal points defined by: top left, top right, left top, left bottom, bottom right, bottom left,right top and right bottom. *Profiles* shape descriptor is based on the number of pixels that the shape has in a given direction: either vertical, horizontal or diagonal. *Moments* refer to the calculation of *moment* statistical property and *profile moment* a combination of the last two.

[22] classifies shape descriptors by their invariance with respect to the transformations allowed in the associated shape definition. The main classes are descriptors invariant with respect to congruency and invariance with respect to isometry. The congruency class comprehends same shape descriptors for congruent shapes (shapes obtained from translation, rotation or mirroring). The intrinsic shape descriptor refer to those that do not change with different isometric embeddings of the shape, and thus can be applied accurately to

deformable objects.

Depending on the properties that are controlled and measured the descriptor may allow or not to reconstruct a shape of the class. In [18] a train-able method of shape representation is described which can automatically capture the invariant properties of a class of shapes and provide a compact parametric description of variability. The method was applied on worms, obtaining a shape descriptor that reconstruct different bending worm shapes by modifying the values of the parameters.

2.8 Cardinal Splines

2.9 Triangle mesh and rasterization

Chapter 3

Methodology

3.1 Development Methodology

3.2 Solution Methodology Design

This sections presents the solution methodology which consists on the different steps in image processing that must be performed in order to successfully fit the shape of *C.Elegans* worms present in digital images. First a general description of the solution is presented where the shape fitting approach is explained, indicating the different processes involved in the solution design. Then for each process a reasoning of its requirement and usefulness is given, as well as the corresponding implementation approach.

3.2.1 Previous reasoning

As stated in [11] and also covered in [19, 20, 21] shape matching is usually accomplished adopting a shape descriptor and then placing a constructed shape sufficiently close to the image shape and adjusting the values of the parameters of the descriptor until a match is found. A shape descriptor is a representation of a specific class of objects that is defined in geometrical terms. It is comprised of a number of parameters, where different values for each parameter give different shapes of a given class of objects. This approach is appropriate when the objects that are to be matched can be categorized in a certain class and thus can be represented or described in terms of geometry, *i.e.* a shape descriptor.

The problem of study aims the detection of worms, particularly those belonging to the *C.elegans* species. Given the vermiform (worm shape) property of these individuals, the objects to detect can be defined as part of a *worm* class, that would refer geometrically to long, thin and cylindrical shapes, in general terms. Following this idea, a shape descriptor could be designed that comprehends a cylindrical, long and connected shape, with two endpoints and different thickness along a medial axis. Then the problem would be reduced to find every pair of endpoints belonging to each worm in the image, place an approximated shape (built through the shape descriptor) near to the matching worm, and adjust the values of the parameters of the shape descriptor until a acceptable match is found.

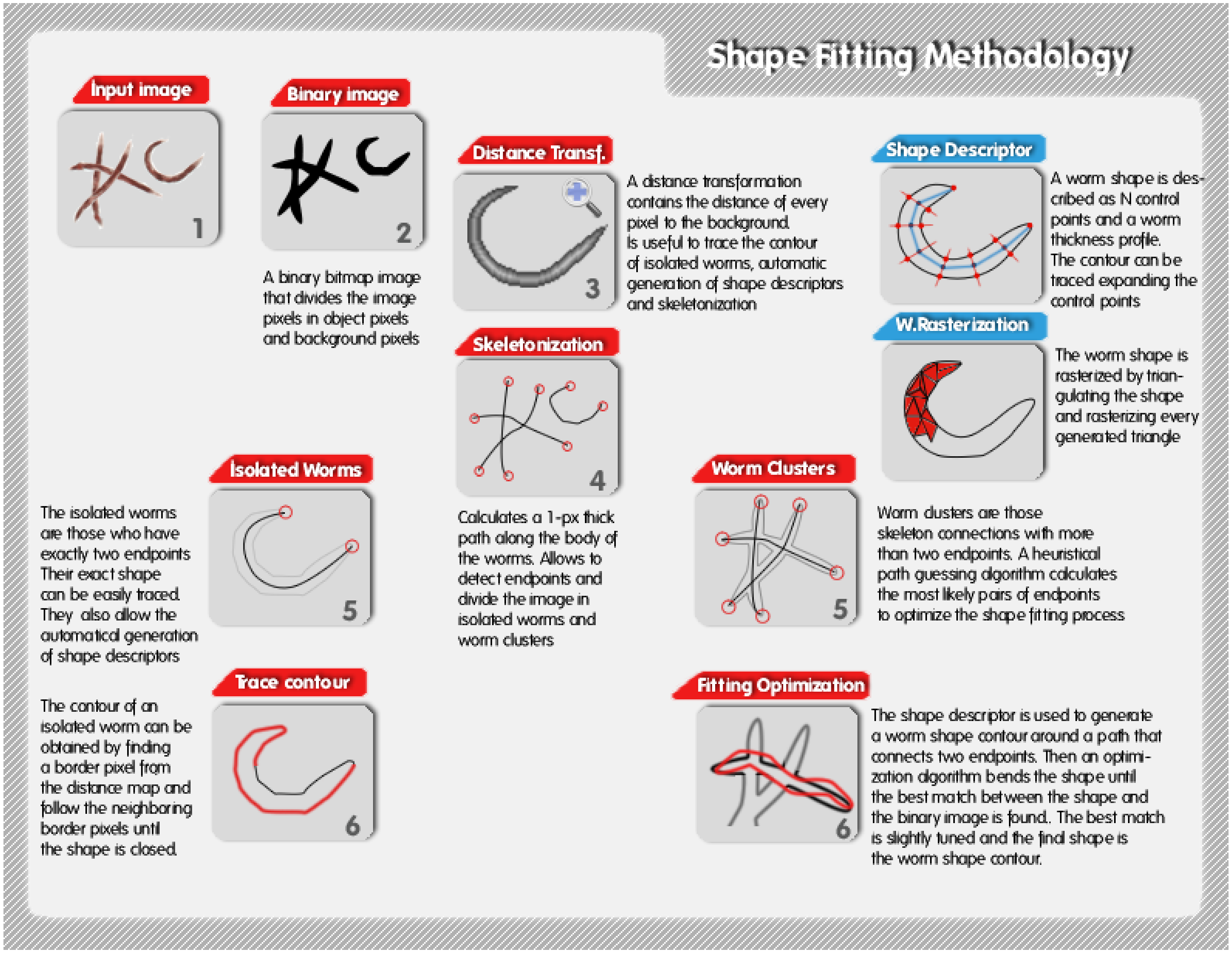
To design a methodology for the solution of the problem the following points must be taken into consideration: the nature of the input images, the positional identification of worms in the global image, the gathering and lost of information and the efficacy and efficiency of each of the involved processes (and their respective algorithms). For this study the input images consist of a number of worms that are put together in liquid media. The image can contain some noise such as shadows, water bubbles or little remains that do not belong to the worms, so these last, as objects of study, must be separated from the rest of the information in the image. The position of each individual worm in the image is variable and can be distinguished into two groups: *worm clusters* and *isolated worms*. A *worm cluster* corresponds to a group of worms in which each worm is connected to any other directly or indirectly through overlapping. It can also be described as a group of worms in which a path can be traced from every worm to another without passing over background pixels. On the other hand, an *isolated worm* is such that is surrounded by background pixels and that does not overlap with any other worm. The image can be segmented by separating the different *worm clusters* and *isolated worms*, so each segment can be processed individually. The contour of *isolated worms* can be traced automatically following the pixels that are closest to background pixels. These can also be used to generate a profile that will set the general values for the shape descriptor that best represents the shape of the worms in the image. The worms that are clustered can be matched through an energy minimization process, based on the manipulation of the shape descriptor and its distance to the matching image, in order to obtain the best possible match.

3.2.2 Methodology description

Following the previous reasoning, a methodology was designed taking into account the main components of the matching process, as reasoned before, such as: determination of the objects of study, objects segmentation, worm shape descriptor and matching based on energy minimization. Below it is described the solution methodology, pointing out the sequential steps that are follow to match and fit the shape of every worm in the input image. In Fig. 3.1 a graphical description of the solution methodology is presented.

Given the input image the first step is to separate the pixels belonging to the object of study from the rest of the image pixels. A thresholding algorithm is then used to obtain a binary image that separates worm pixels from background pixels. Usually some noise is obtained after the thresholding process, but it is ignored in further processing. This corresponds to an initial segmentation of the input. Once had a binary image, a distance map can be obtained in which each pixel represents the distance to the nearest background pixel, as explained on [4]. The distance transformation allows to identify the contour pixels in the binary image, which makes it a fundamental tool for the automatic generation of a shape descriptor, contour tracing on isolated worms and optimization of the skeletonization algorithm, among others. Having the image separated in object pixels and background pixels the image can be segmented to separate the worms.

The image worms have been distinguished into two groups: *isolated worms* and *worms clusters* (see Sec.3.2.1). A way to differentiate the image worms is to count the number of endpoints of every object (as defined after the binary transform). An object with exactly two endpoints would correspond to an *isolated worm* while more that two points indicates the presence of overlapping worms, thus a *worm cluster.* At the same time a path from one endpoint of a worm to another is needed to match the shape, due to the need of placing the matcher shape near to the matching shape, which is a usual approach for shape matching, as it was covered on Sec. 2.6 and Sec. 3.2.1. Having said this, a process of skeletonization would provide a simple way to recognize endpoints as well as an approximated connected medial path between the endpoints of the image objects. After this process, the image will be segmented into many different subareas of the images corresponding to both clusters and *isolated worms.* Each type of worm group is processed in a different way in order to match the shape.



A binary bitmap image that divides the image pixels in object pixels and background pixels

A distance transformation contains the distance of every pixel to the background. Is useful to trace the contour of isolated worms, automatic generation of shape descriptors and skeletonization

A worm shape is described as N control points and a worm thickness profile. The contour can be traced expanding the control points

The worm shape is rasterized by triangulating the shape and rasterizing every generated triangle

The isolated worms are those who have exactly two endpoints Their exact shape can be easily traced. They also allow the automatical generation of shape descriptors

Calculates a 1-px thick path along the body of the worms. Allows to detect endpoints and divide the image in isolated worms and worm clusters

Worm clusters are those skeleton connections with more than two endpoints. A heuristical path guessing algorithm calculates the most likely pairs of endpoints to optimize the shape fitting process

The contour of an isolated worm can be obtained by finding a border pixel from the distance map and follow the neighboring border pixels until the shape is closed.

The shape descriptor is used to generate a worm shape contour around a path that connects two endpoints. Then an optimization algorithm bends the shape until the best match between the shape and the binary image is found. The best match is slightly tuned and the final shape is the worm shape contour.

Figure 3.1: Graphical description of the solution methodology for the shape detection of C.elegans worms in digital images

Isolated Worms

The *isolated worms* shape contour can be traced easily by selecting a border pixel (indicated in the distance map) and following the neighboring contour pixels until the initial pixel is reached back again. Then the whole shape can be rasterized by triangulating it through the *ear clipping triangulation method* and then rasterizing each triangle separately. This provides all the pixels belonging to the shape, *i.e.* a match. The nearly perfect match that can be obtained from the *isolated worms* makes possible to calculate a worm profile from the currently analized worms in order to generate an accurate shape descriptor, this is explained thoroughly in Sec. 3.2.7.

Worm Cluster

To match the shape of the worms present in a *worm cluster* the methodology consists in determining the number of worms in the cluster (follows from the number of endpoints) and finding the best match between pairs of endpoints. Given a pair of endpoints the path between them is calculated and then a matcher worm shape is generated from the shape descriptor, selecting a given number of control points in the path. Then an energy minimization process is performed that varies the angles between the straight lines that connect the control points (generating different shape representations), until the best match is found. Then the contour of the shape descriptor is slightly modified by finding the closest contour segments (or a lower value in the distance map) to the contour points, in order to adapt the generic shape silhouette to the matching object.

This process can be repeated between every pair of endpoints and then choose the best matches. On the other hand a path guessing algorithm can be used to find the most likely path between endpoints, *i.e.* the path from one point to another in the cluster skeleton that most likely belongs to a worm (see. **??**), in order to avoid the cost of trying every pair of endpoints.

In the following sections, each of the subprocesses involved in the solution methodology are explained, covering their need and usefulness as well as the followed implementation approach.

3.2.3 Thresholding

Since the main purpose of this study is to fit the shape of *C.Elegans* worms on digital images, it is useful to differentiate these from the rest of the image in order to perform a more accurate analysis. The shape of the worms can be characterized as objects and the rest of the image as background. More precisely the image pixels can be separated into two groups: object pixels, that are all of those that belong to a worm shape and background pixels, that are all the remaining ones.

Given this theoretical characterization, a thresholding filter would come to be a useful tool to locate the objects of study in the digital representation and to discard unnecessary information, obtaining a binary image from the original one. A binary image would then provide an initial segmentation of the processed image, being as well a key element to obtain a distance transformation, as it is explained in Sec. 3.2.4.

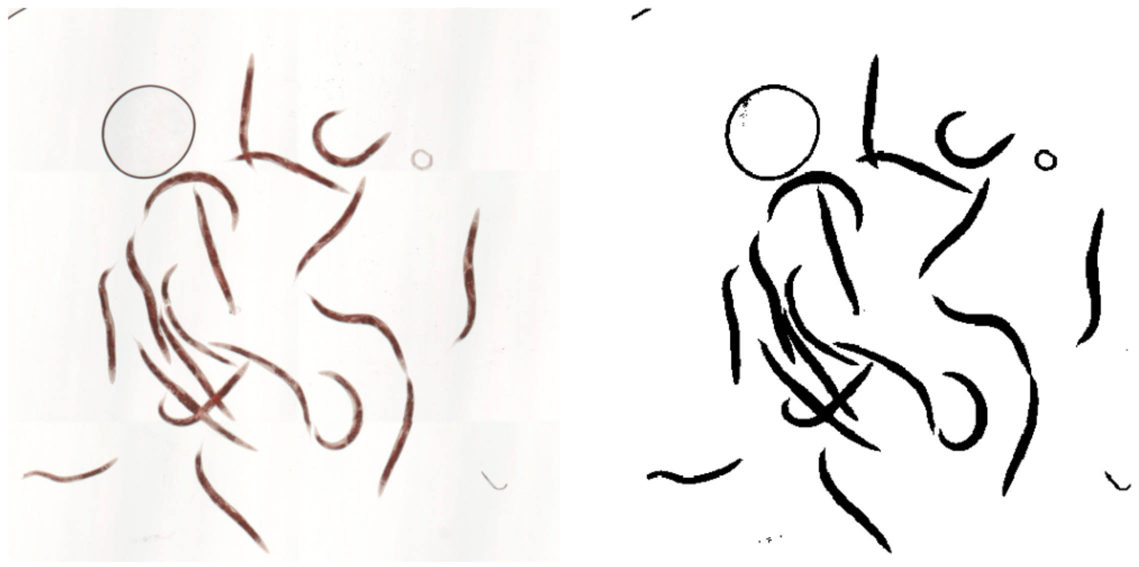
3.2.3.1 Implementation

There are four thresholding filters for 2D images implemented on *Endrov*, these are: *Fukunaga*,*Max entropy*, *Otsu* and *Percentile*, that cover the histogram and entropy-based thresholding methods categories as defined in Sec.2.2. Considering that the implemented methods are sufficiently different and given the transparency of *C.Elegans* worms is hard to determine theoretically which would be the most appropriate thresholding method to obtain an accurate binary image, from the study data-set. In order to select a thresholding method a series of experiments where performed tweaking the parameters for the different mentioned methods, as it is explained on Sec.**??**. The selected method was *Percentile Threshold 2D* with a percentile value oscillating from 0.072 to 0.09 on the different test images.

Figure 3.2 shows a binary image obtained after applying the *Percentile Threshold 2D* method with a percentile value of 0.074

3.2.4 Distance transformation

In this shape fitting approach for *C.Elegans* worms the distance transformation of the given image is used thoroughly for contour detection and different kinds of image segmentation procedures. Specifically the distance map allows to detect and follow the exact contour of



(a) Original image (b) Percentile Threshold. Value=0.074

Figure 3.2: Worms in liquid media original image and binary image obtained through Percentile Thresholding with a percentile value of 0.074

isolated worms (Sec. 3.2.9.2), is useful in the shape profile generation (Sec. 3.2.7.1), and essential in the heuristic guessing of the more likely worm-paths on *worm clusters* (Sec. ??). It also improves the performance of the iterative thinning algorithm designed by *Zhang and Suen* [26] as it is described on Sec. 3.2.5

3.2.4.1 Implementation

As stated in [25, p.196] the algorithms of DT can be categorized into two classes: one is the iterative method which is efficient in a cellular array computer since all the pixels at each iteration can be processed in parallel, and the other is sequential (or recursive) method which is suited for a conventional computer by avoiding iterations with the efficiency to be independent of object size. Using the general machines that most people working in digital image processing have access to, sequential algorithms are often much more efficient than iterative ones. For this reason a sequential approach was chosen to calculate the distance transformation of the input images. Particularly the two-scans transformation using 3x3 neighborhoods [25] which is both efficient and easy to implement.

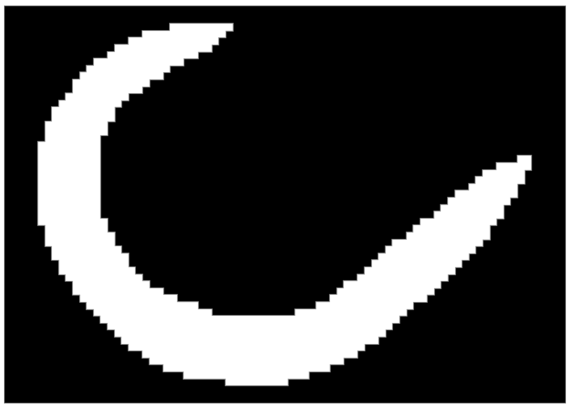
In the mentioned paper a distance map calculation algorithm is described which consist on only two scans of the image bitmap, one left to right - top to bottom, and another right to left - bottom to top, with one operation per pixel. This makes the complexity of the algorithm $\mathcal{O}(N)$ for N the size of the image array. In [25, p.197] a pseudo-code for *Chessboard* and *Manhattan or city-block* distances is given, while in [25, p.198] the definition is extended to improve the efficiency of the calculations needed to generate a distance map using *Euclidean* distances. The two-scans algorithm was implemented using the three different distance metrics mentioned before. This allows a wider analysis on the behavior and accuracy of the shape fitting process from one metric to another.“The city block or chessboard distance measures are sensitive to the rotations of an object, but the Euclidean distance measure is rotation invariant. However, its square root operation is costly...” [15, p.332]. Given the straight-like shape of worms and the different levels of accuracy of the distance metrics it is hard to tell at first sight which would be the most adequate to use, so it had to be determined experimentally, as it is explained on **??**. The figure 3.3 shows the binary image and three distance maps obtained from a single worm image.

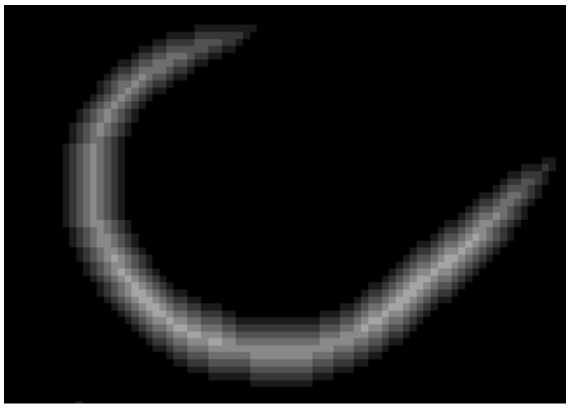
3.2.5 Worm Skeletonization

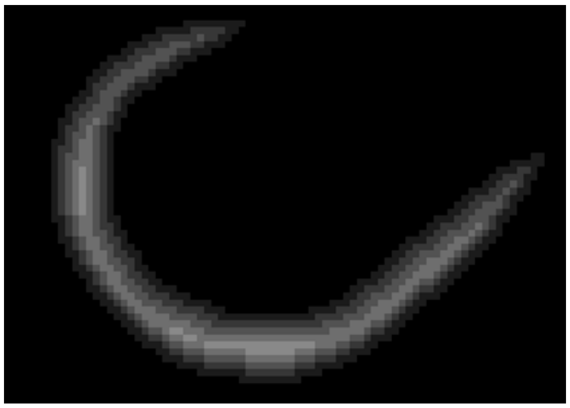
The skeletonization of the image corresponds to the process of obtaining a connected and thin (1-pixel width) medial axis that represents the worms in the image. This is a key process on the shape matching approach followed in this work, as first mentioned on Sec. 3.2.2. It allows to identify the amount of worms in the input image, to separate them distinguishing between *isolated worms* and *worm clusters*, and to obtain an approximated path between endpoints of worms (that tends to the medial axis), which is the main element of the matching optimization process (see Sec.3.2.6). The skeleton image would be then fundamental to determine the area of the image in which the worms are located and give estimated paths along which the different worms would be disposed.

3.2.5.1 Implementation

For the purpose of this work the skeletonization algorithm to be selected must ensure the connectivity of the skeleton points, *i.e.* every skeleton point must be connected to at least another skeleton point belonging to the same skeleton. Also the skeleton connection must


(a) Binary Image


(b) Manhattan metric


(c) Chessboard metric

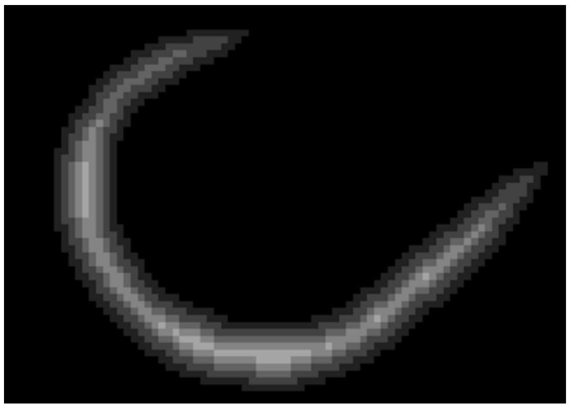

(d) Euclidean metric

Figure 3.3: Binary Image and Three Distance Transformation metrics from a single worm image

not be thicker than 1-pixel, this means that when tracing a sequential path in the skeleton and being situated in a given skeleton point there must be only one neighbor skeleton point that can be selected to continue the path direction, while remaining on the track of the skeleton of the same worm.

Provided that a distance map have been already calculated when the skeletonization process is carried out, an efficient approach would be to use a distance transform based technique. Different methods consisting in finding ridge points on the distance maps and then connect them have been covered in [16, 2, 3]. The approach in [16] was followed as a first attempt to calculate a thin skeleton with a low time cost. This algorithm defines the ridge points as such pixels that have the greatest numerical value among its 3x3 neighborhood (bitmap image) in the distance map. After finding the ridge points a *up-hill* reconnection is performed, followed by a *down-hill* connection for missing points. The study covered in [16] states that this approach allows to find successfully a connected 1-pixel-thin skeleton, which was actually the case for *isolated worms* or those who do not overlap with other worms. Nevertheless for *worm clusters* the obtained skeletons were usually disconnected, thicker than 1-pixel and not accurate. Although the approach seemed appropriate in theory, the costful reconnecting operations and the inaccurate skeletons obtained for *worm clusters* gave rise to the need for a different approach.

Given the long, thin and cylindrical shape of the worms in general, a thinning algorithm approach that reduces the different layers by removing pixels that should not belong to the skeleton was then taken into account. In [26] an iterative and parallel thinning algorithm is presented, that consist in two sub-iterations per main iteration aimed at deleting the south-east boundary points and north-west boundary points respectively. The study is aimed at parallel computers so the different operations in each pixel can be performed at the same time, improving the performance. In order to avoid the requirement of using a parallel computer without losing the time performance improvement, the distance map was used to discard unnecessary pixel checking (those belonging to inner layers). Thus in each iteration only the pixels belonging to the currently selected shape layer are taking into account. The layers are defined by the distance map value of its pixels. The first layer corresponds to a distance value of one (1), the next to a distance value of two (2) and so on.

Below in algorithm 3.2.1 is presented a pseudo-code version of the thinning algorithm covered in [26] that follows the distance transformation layers approach.

The algorithm deals well with overlaping worms by constructing a path that is very close from the shapes medial axis, and results in a totally connected 1-pixel width skeleton. In fig. 3.4 the skeleton of a sample image is shown. Every worm in the image that was previously identified in the binary image obtaining process (see 3.2.3) is successfully *skeletonized*.

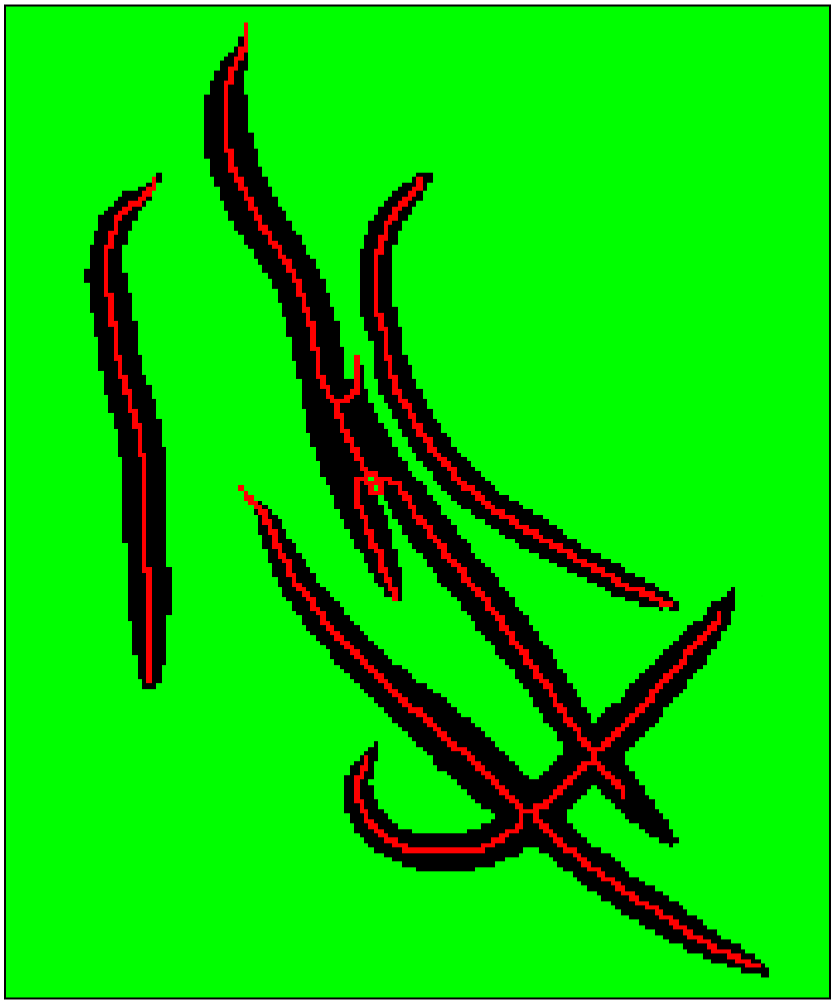


Figure 3.4: Skeleton obtained through iterative thinning over a worm binary image

Algorithm 3.2.1 Calculate shape skeleton

```
$shapePts \leftarrow getBinaryObjectPixels()$
$dtImage \leftarrow getImageDistanceMap()$
$contourIndex \leftarrow 1$
$makeThinner = True$
while $makeThinner$ do
    {remove south-east boundary points and the north-west corner point}
    for $pixel$ in $shapePts$ do
        if $dtImage(pixel) > contourIndex$ then
            {skip iteration}
        else
            $pixelRemove \leftarrow southEastCondition(pixel)$
            if $pixelRemove$ then
                $shapePts.remove(pixel)$
                $makeThinner \leftarrow True$
            end if
        end if
    end for
    {remove the north-west boundary points and the south-east corner points}
    for $pixel$ in $shapePts$ do
        if $dtImage(pixel) > contourIndex$ then
            {skip iteration}
        else
            $pixelRemove \leftarrow northWestCondition(pixel)$
            if $pixelRemove$ then
                $shapePts.remove(pixel)$
                $makeThinner \leftarrow True$
            end if
        end if
    end for
end while

return $shapePts$
```

3.2.6 Worm Segmentation

Since the approach of the study aims to match the shape of the different worms individually, it is necessary to locate them in the image, and then separate them as much as possible, *i.e.* to segment the image (see Sec. 2.5). This allows to improve the efficiency and accuracy of the shape fitting process, while reducing the matching area and thus the amount of different combinations that must be taken into account. After the process of *skeletonization* a set of paths are obtained between endpoints of worms (the objects of study), in some cases overlaping. By identifying worms endpoints and then tracing the paths that connects them together, the different groups of paths can be separated, thus segmenting the image. As explained in Sec. 3.2.5 and Sec. 3.2.1, the different groups of paths can be distinguished, in correspondence to the objects they represent, by *isolated worms* and *worms clusters*. Then the shape fitting process of the whole image would consist in matching and fitting the worms present in each of the obtained sub-images separately.

Another process of segmentation that is performed is the identification of single worms paths, in both *isolated* and *cluster* worms. For *isolated worms* the path that determines its medial axis is used, first to find the surrounding contour to fit its shape (see Sec. 3.2.9.2), and second to generate a profile that would define a general representation of the worms in the image through a shape descriptor, as it is explained in Sec. 3.2.7.1. On the other hand, for *worm clusters*, paths must be found between endpoints that permit the construction of a general shape, in order to determine if a worm can be matched between those two endpoints through the matching optimization process. These paths can be simply found by tracing all the possible “roads” between endpoints, or by a more sophisticated path guessing algorithm that will be described below in the implementation section.

3.2.6.1 Implementation

Worm Endpoints

The skeleton calculation returns an image containing 1-pixel-width groups of curves or paths, which means that each pixel is neighbor of either two other pixels or one. The pixels that are connected with two other pixels (two neighbors) are *body-pixels*, which are such that belong to the path and are not endpoints. On the other hand those that are connected with only one other pixel are endpoints, thus each one could belong to the extreme of a worm.

It is important to consider that since the thinning algorithm used to obtain the skeleton (covered in Sec. 3.2.5.1) consists in removing the shape layers until finding pixels that are not surrounded, the endpoints found in the skeleton will not necessarily correspond to worm endpoints.

In order to find the pixels that are actually worm endpoints a skeleton expanding process is performed that aims to stretch the skeleton path up to a contour point that would come to be the worm endpoint. The skeleton expanding algorithm uses the definition of *directional neighborhood* stated in [16, p.334], where given a pixel P in the bitmap, a *directional neighborhood* D of P is conformed by those pixels that belong to the *8-neighborhood* of P and that are located within ± 45° slope changes from the current medial axis orientation of P. Below in fig. 3.5 three examples of directional neighbors are presented. The algorithm consist in following the best directional path starting in every endpoint and expanding the skeleton until a contour pixel is found.

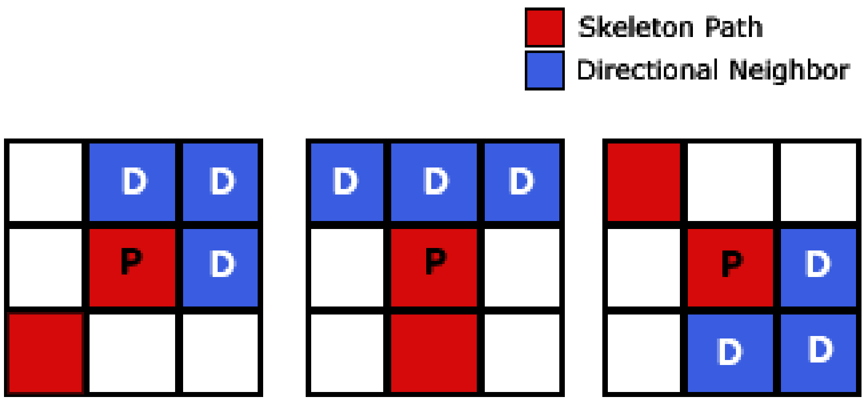


Figure 3.5: Three different directional neighborhoods

The expanding algorithm can be summarized in the following steps:

- Select an endpoint
- Find the previous skeleton point and calculate the directional neighborhood
- Select the directional neighbor with the lowest distance transformation pixel, and mark it as skeleton pixel.

- If the neighbor is not a contour point repeat the process.

A filtering process is done to remove incorrect object pixels, that consists in removing sub-skeletons that have a size (number of pixels) lower than a small threshold. This allows to remove some slightly noisy regions and incorrect endpoints. Once the skeleton is successfully expanded, the endpoints of the skeleton are considered worm endpoints. It must also be considered that some worm endpoints cannot be detected following this process, particularly in crowded images where there is a big chance that the overlaps "hide" the endpoints. This is considered when selecting paths between endpoints, as explained in Sec. 3.2.6.1.

Group segmentation

Having detected every worm endpoint the image can be segmented by determining the endpoints that are bound together through a skeleton path, *i.e.* finding the different *isolated worms* and *worm clusters*. As it was previously explained, the overlaping worms in the image are bound together as one object in the binary image obtaining process, so then in the skeletonization process the endpoints belonging to *worm clusters* are bound together as well, through a skeleton path. Based on the previous reasoning an algorithm was designed that detects the endpoints that are bound together through the path that links them, separating then the linked paths in different groups. Those paths that link together exactly two endpoints represent *isolated worms* while a different number of endpoints correspond to *worm clusters*. Below in algorithm 3.2.2 a pseudo-code for this group segmentation method is presented. The algorithm consist basically following every different possible path starting on an endpoint until all the endpoints of a group have been reached.

Algorithm 3.2.2 Calculate shape skeleton

```
$endPtList \leftarrow$ list of endpoints
$clusterIndex \leftarrow 0$
for $endpoint$ in $endPtList$ do
  if $endpoint.wasVisited()$ then
    {skip iteration}
  else
    $clusterIndex += 1$
    $followPath(endpoint, clusterIndex)$
  end if
end for
```

Algorithm 3.2.3 Follow Path algorithm ($followPath(currentPoint, clusterCount)$)

```
Require: $currentPoint$
Require: $clusterCounter$
  if not $currentPoint.isSkeletonPoint()$ then
    return
  else
    $addToCluster(endpoint, clusterIndex)$
  end if
  {continue tracing path}
  if $currentPoint.isEndPoint()$ then
    $markEndPointAsVisited(currentPoint)$
  end if
  $neighbors \leftarrow getNeighborhood()$
  for $n$ in $neighbors$ do
    $followPath(endPoint, clusterCounter)$
  end for
```

Path guessing

The *worm clusters* found through segmentation are defined by a set of endpoints that are all connected through *skeleton* paths, however the pair of endpoints belonging to each worm in the image and an accurate skeleton path that connects them is still unknown. The optimization algorithm, covered in 3.2.9, performs a shape manipulation process to match the worms in the image given two endpoints and the path that connects them. When the right skeleton paths are unknown the algorithm tries every possible combination returning the best match as possible, with the consequent overhead in performance time . In order to improve the time efficiency of the algorithm as well as the accuracy of the matching (particularly avoiding incorrect endpoints bounding), a path guessing algorithm was designed that performs a heuristic guessing of the most-likely path between endpoints.

The algorithm is based in the idea of avoiding paths that continue unnatural conformations of the worms shape. This means that given S last steps in the skeleton tracing, the next step $S + 1$ will tend to follow the direction that was being followed, thus avoiding unnatural bendings or abrupt changes in the shape. An important issue when following the most common direction in the last S steps is that in some cases the path tracing will tend to avoid *path bifurcations*. A *path bifurcation* occurs when there is more than one neighbor pixel that can be followed next, thus dividing the path in two or more different paths.

In order to make the path tracing tend to reach path bifurcations and then decide the best path to take, a heuristic function was designed that consists in the value of the neighbor pixel in distance transformation multiplied by a variable factor. Then the selection of the next pixel to follow its based in two main values: the amount of time that the neighbor direction has been taken in the last S steps and the distance map value of the pixel. This can be expressed as below:

$$Next(p) = \max_{n \in neighbors(p)} (dirValue(direction(p, n), S) + dt(n) * hfactor)$$

where p is the current path pixel, $dirValue$ returns the amount of times that the direction of the neighbor pixel p have been taken in the last S steps, dt is the distance map and $hfactor$ is a heuristic factor that controls the influence of the distance map.

In order to cover the issue mentioned in Sec.3.2.6.1, that some endpoints could be hidden because of the overlapping, the number of pixels that have been added to a tracing path are counted, so when it is not possible to reach any other endpoint, an extra endpoint is created after *wormLength* steps. The *wormLength* is an estimated value that is calculated as the mean of the length of the *isolated worms* in the image. This process attempts to identify worms that otherwise would be discarded because of the absence of endpoints. Below, algorithm 3.2.4 presents a pseudo-code for the path guessing approach.

3.2.7 Worm Shape Descriptor

As first mentioned on Sec. 3.2.2 the selected methodology approach to match *C.Elegans* worms is based on the manipulation of worm shapes generated from a shape descriptor. Worm shapes can be described in geometrical terms as long, thin and cylindrical objects. Given that the process of skeletonization and image segmentation allow to obtain *thin* paths between pairs of endpoints that correspond to worm extreme points, a shape descriptor would permit to construct a shape around this medial axis that would work as a starting point for the optimization algorithm.

A shape descriptor was then designed based on the idea of generating a representative worm shape around the medial axis. The descriptor consists in two elements: control points and shape profile (worm profile). The control points are a set of N equidistant points

Algorithm 3.2.4 Pseudo-code algorithm for path guessing between endpoints

```
$endPtList \leftarrow$ list of endpoints
$wc \leftarrow$ paths and endpoint in worm cluster
$length \leftarrow$ worm estimated length multiplied by a scaling factor
for $endPoint$ in $endPtList$ do
  if $alreadyReached(endPoint)$ then
    {skip iteration}
  end if
  $markAsReached(endPoint)$
  $path \leftarrow$ empty list
  $reachedEndPoint \leftarrow False$
  $currentPixel \leftarrow endPoint$
  while $not(reachedPoint)$ and $size(path) < length$ do
    $currentPixel \leftarrow getBestNeighbor(currentPixel)$
    $updateDirectionsArray(direction(currentPixel))$
    $path.add(currentPixel)$
    if $isEndPoint(currentPixel)$ then
      $reachedEndPoint \leftarrow True$
    end if
  end while
  if $not(reachedEndPoint)$ then
    if number Of Reachable Endpoints $= 0$ then
      $createEndPoint(currentPixel)$
      $path.add(currentPixel)$
    else
      {Select a path to reachable endpoint}
    end if
  end if
end for
```

along the worm medial axis, both endpoints included. Each control point has associated a *thickness* value that represents the radius of a circle with the control point as center. Then by selecting two points in opposite directions of the thickness circle for every control point, and joining these points together through a smooth curve, a contour line is obtained that traces the silhouette of a worm shape, as shown in fig. 3.6. The set of *thickness* values that are associated to each control point is called shape profile (worm profile for this study).

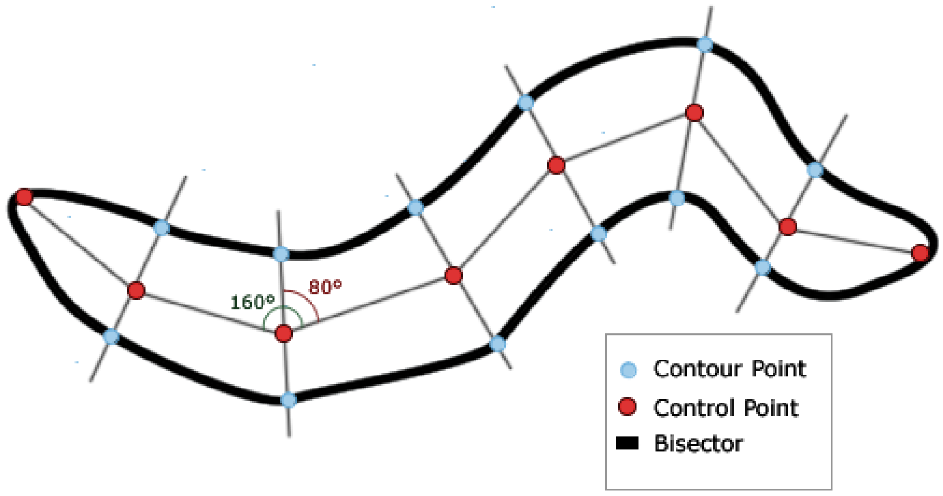


Figure 3.6: Construction of a worm shape based on shape descriptor

In order to obtain a contour that accurately represents the worm shape along the medial axis, the choice of the opposite points in the *thickness* circles must take into account the *skeleton* bendings. Since the shape is constructed following the *thickness* of control points, the bendings in the generated shape occur at each of these ones, and can then be calculated as the angle between the straight lines that connect the control points. The bisection of any angle provides a straight line that divides it into two equal portions, measuring the bending variation at each control point. Then selecting the two points at a respective *thickness* distance, from every control point, following the bisection line in opposite directions provides a set of points that joined together with a smooth curve will give a worm shape contour.

Generating a smooth curve around the control points improves the accuracy of the shape description, in comparison with an approach consisting in tracing straight lines between the contour points. At the same time it allows to obtain a good representation of the shape with a considerably smaller number of control points. The smooth curve is obtained by calculating the cardinal spline (covered in Sec. **??**) given the contour points. A cardinal spline is a function that describes a smooth curve that passes over all the points of a set, given a starting and ending point. In this case the starting and ending points are the same, so the described contour is closed.

The worm profile for a set of control points can be both manually set or automatically calculated from the *isolated worms* as explained in the section below.

3.2.7.1 Automatic Profile generation

The shape of *isolated worms* identified in the image can be accurately matched by following the contour points in their respective distance map, as explained in Sec. 3.2.9.2. Given a set of matched shapes for *isolated worms*, and the initially detected skeleton, a worm profile (as described previously) can be generated by measuring the *thickness* of every control point and finding the mean among these.

In order to measure the thickness for every control point, a set of N equidistant points is generated that cover the skeleton of the *isolated worm*. Then as described in the previous section, the bisectors of the angles between the straight lines connecting the control points are calculated. Starting from every control point, the bisector line is "walked" until the pixel with the lowest distance map value is found, a contour point in most of the cases. The operation is repeated for every control point bisector in both opposite directions. The Euclidean distance from a control point to each found pixel is calculated, and its average is recorded. Repeating this process with every *isolated worms* generates a set of *thickness profiles*, one for each *isolated worms*. Then a general worm profile is calculated by finding the mean between the thickness values for every control point in each *thickness profile*. In order to generate an accurate profile, that avoids oversized (or downsized) worms and wrongly detected *isolated worms*, the 20% higher and lower values are discarded when calculating the average. The *thickness* value for the endpoints, *i.e.* the first and last point in the set of length N, is zero, so one contour point is generated in the extremes instead of two. Having done this a *thickness profile* is obtained that represents the average radius distance for every control point to its closest contour pixel, allowing to generate a generic worm shape around any skeleton defined by N points.

3.2.8 Triangle mesh and rasterization

3.2.9 Profile-driven shape fitting

The implementation is being modified, so this part will be completed later

3.2.9.1 Worm Cluster shape fitting

3.2.9.2 Isolated Worms shape fitting

Chapter 4

Experiments and Results

Chapter 5

Conclusions

Bibliography

[1] Carlo Arcelli and Gabriella Sanniti di Baja. Ridge points in euclidean distance maps. *Pattern Recognition Letters*, 13:237–243, 1992.

[2] Carlo Arcelli and Gabriella Sanniti di Baja. Euclidean skeleton via centre-of-maximal-disc extraction. *Image and Vision Computing*, 11:163–173, 1993.

[3] Carlo Arcelly and Gabriella Sanniti di Baja. Ridge points in euclidean distance maps. *Pattern Recognition Letters*, 13:237–243, 1992.

[4] Gunilla Borgefors. Distance transformation in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[5] Roland T. Chin and Hong-Khoon Wan. A one-pass thinning algorithm and its parallel implementation. *Computer Vision, Graphics and Image Processing*, 40:30–40, 1987.

[6] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance transforms of sampled functions. *Cornell Computing and Information Science Technical Report*, 2004.

[7] Rolf Lakamper Longin Jan Latecki and Ulrich Eckhardt. Shape descriptors for non-rigid shapes with a single closed contour. *Computer Vision*, 1:424–429, 2000.

[8] Xiang Bai Longin Jan Latecki, Quan-nan Li. Skeletonization using ssm of the distance transform. Technical report, Temple University, Philadelphia, USA, HuaZhong University of Science, and Technology, Wuhan, China, 2007.

[9] Andrew Witkin Michael Kass and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–333, 1988.

[10] Bryan S. Morse. Shape description (regions). http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL props-and-moments.pdf.

[11] P.E.Trahanias. Binary shape recognition using the morphological skeleton transform. *Pattern Recognition*, 25:1277–1288, 1992.

[12] Matthew Baker Sasakthi Abeysinghe, Tao Ju and Wah Chiu. Shape modeling and matching identifying 3d protein structures. *Computer-Aided Design*, 40:708–720, 2008.

[13] Mehmet Sezgin and Bulent Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging*, 13:146–165, 2004.

[14] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, 2002.

[15] Frank Y. Shih and Christopher C.PU. A skeletonization algorithm by maxima tracking on euclidean distance transform. *Pattern Recognition*, 28:331, 1994.

[16] Frank Y. Shih and Christopher C. Pu. A skeletonization algorithm by maxima tracking on euclidean distance transform. *Pattern Recognition*, 28:331–341, 1995.

[17] Alexandru Telea and Jarke J. van Wijk. Shape descriptors for non-rigid shapes with a single closed contour. In *EUROGRAPHICS - IEEE TCVG Symposium on Visualization*, 2002.

[18] C.J.Taylor T.F.Cootes, D.H.Cooper and J.Graham. A trainable method of parametric shape description. *Computer Vision*, 5:237–252, 2009.

[19] Marcel Luthi Thomas Albrecht and Thomas Vetter. Deformable models. gravis.cs.unibas.ch/publications/CH_Deformable_Models09.pdf.

[20] Remco C. Veltkamp. Shape matching: Similarity measures and algorithms. Dept. Computing Science, Utrecht University.

[21] Remco C. Veltkamp and Michiel Hagedoorn. State-of-the-art in shape matching. *Principles of visual information retrieval.-(Advances in pattern recognition)*, 2:87–112, 2001.

[22] Wikipedia. Shape analysis. http://en.wikipedia.org/wiki/Shape_analysis.

[23] Wikipedia. Topological skeleton. http://en.wikipedia.org/wiki/Topological_skeleton#cite_note-8.

[24] Wikipedia. Thresholding(image processing). `http://en.wikipedia.org/wiki/Thresholding_(image_processing)`, June 2010.

[25] Frank Y.Shih* and Yi-Ta Wu. Fast euclidean distance transformation in two scans using a 3 x 3 neihborhood. *Computer Vision and Image Understanding*, 93:195–205, 2002.

[26] T.Y. Zhang and C.Y. Suen. A fast parallel algorithm for thinning digital patterns. *Image Processing and Computer Vision*, 27:235–239, 1984.