Local Method of Length and Direction Estimation for Stepped Representations of Curves

Application to the 'Weak Membrane' Image Restoration Algorithm

Leszek Chmielewski
EPSILON *Applied Research Group Co.Ltd.*
Daniłowiczowska 11/66, PL 00-084 Warsaw

May 1993

Abstract A simple method for retrieving local length and orientation of a curve represented as a set of edge elements (inter-pixel boundaries) is proposed. This entirely local method is based on classifying the layout of an edgel and its closest neighbours and resolves itself to referencing a small look-up table. No "smooth" representation such as a fitted polynomial is used. Simple examples as well as an advanced application to the enhancement of the 'weak membrane' image restoration algorithm are presented.

1. Introduction

In computer images represented as arrays of square pixels, the curves are frequently represented as strings of pixels, or strings of inter-pixel boundaries (Fig. 1). Curves as loci of edge pixels result e.g. from numerous edge finding algorithms (Prewitt, Sobel, Canny operators etc.). These loci can then be thinned, to obtain one-pixel-wide lines, i.e. strings of pixels. The curves represented by inter-pixel boundaries are less frequently encountered, probably due to their less straightforward representation. However, a line element representing an element of an edge is treated as something located *between* two specified pixels in such methods as filters derived from the Markov random field theory [8, 13] or filters based on the concept of a 'weak membrane' or 'weak plate' [11].

The problem we shall study is the problem of estimation of the real curve length and direction (inclination angle). In an image we have the discrete, quantized representation of a curve, while the real curve is smooth. What will interest us is how to estimate its length and direction locally, i.e. how to guess the length and angle of a fragment of the curve represented by a single line element, on the basis of information provided by the location of only itself and its immediate neighbouring line elements. The locality of our method consists in that both its input and output is the local information.

In the present paper we shall be concerned only with the inter-pixel boundaries, which we shall call the *stepped curves* to reflect the fact that their elements can be only horizontal or vertical, as far as square pixel grids are used in general. Nevertheless, the concept proposed here is applicable also to the 'string of pixels' representation, after some modifications. An interesting discussion on where the curve which represents the boundary of the original blob lies in its digital image can be found in [6]. The

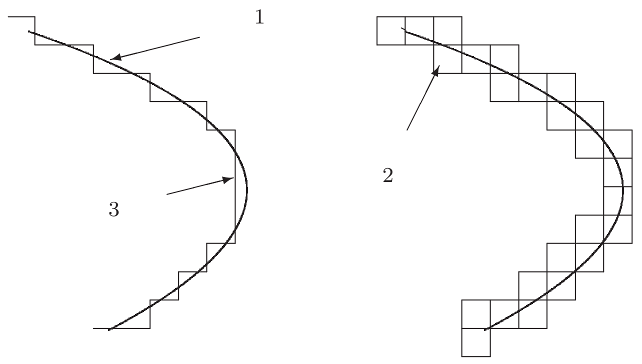


Fig. 1. Physical curve and its representations: 1 – curve; 2 – representation as string of pixels; 3 – stepped representation as string of inter-pixel boundaries

choice of the inter-pixel boundary as the curve location, made in the development of the method described in this paper, was triggered by the challenge of applying this method to improve the performance of the 'weak membrane' image enhancement and edge detection algorithm [11]. That algorithm will serve us as an example of application of our local curve length and angle estimator.

Numerous relevant global methods are already in use; among them we can name polygonal approximation [1, 5, 10] and the related m-sampling method [4, 3], spline fitting [15, 9, 7, 14] or curvature smoothing [3]. In all these methods, in spite of the locations of the discrete line elements, also their sequence must be known.

Two interesting methods of curve length estimation in which the information on the line elements sequence is not important were proposed by Kulpa [2, 6] and Proffitt, Rosen, Ellis and Rutkowski [4, 3]. Kulpa uses the numbers of horizontal, vertical and skewed (inclined at 45°) elements; hence, his method is suited rather to the curves coded with eight directions than to those coded with four ones, as it is in our case. Proffitt et al. considered four-direction coded lines, and their method involves counting the elements and the corners. Both methods are 'partly local' in the sense that they use only local features of the curve representation, and can be implemented with local algorithms; however, the information they provide (the total curve length) is still global.

Neither global nor partly local methods are applicable in local algorithms which necessitate local curve length, as the 'weak membrane' or 'weak plate'.

No reference on estimating the local direction of a digitized curve is known to the author so far.

What we consider here is a pure 'mesh effect' resulting from images being the most frequently represented (approximated) as square grids of image elements.

In images, curves or lines usually represent edges. We shall treat the notions *curve*,

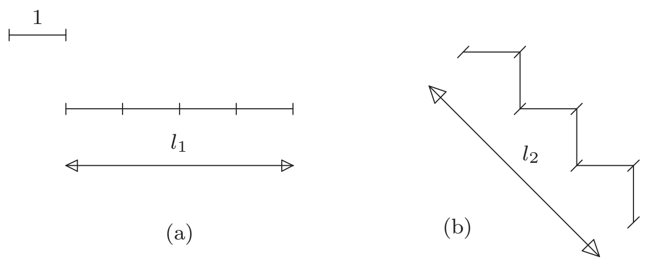


Fig. 2. Length error for stepped representation of a line: (a) horizontal line; real line and stepped line length $l_1 = 4$, zero error; (b) line inclined at 45°; real line length $l_2 = 3\sqrt{2}$, stepped line length 6, relative error $\sqrt{2} - 1$

line and *edge* as equivalent.

The general approach proposed here is the following. Coarse line length estimation consists in mere counting the line elements. If pixels are square, then for horizontal or vertical lines the relative estimation error is zero, but for inclined lines it grows up to $2/\sqrt{2} - 1 = \sqrt{2} - 1 \approx 41\%$, where lines inclined at 45° are the worst case – cf. Fig. 2. As we look at that figure, we tend to perceive the stepped line (b) as a representation of an inclined straight line rather than a stepped one. The length of the straight line, and its inclination angle, is related to the shape of the stepped line. Our local length estimation method will be based on such an intuitive relation.

Each element of the stepped line can be considered equivalent to an element of the inclined one, and, in this sense, each has its equivalent length and direction. In the case shown in the Fig. 2 b this equivalent length is $1/\sqrt{2}$, and the equivalent direction is north-west.

Our problem consists in finding such the equivalent lengths and directions for line elements in lines represented with stepped curves.

2. The Method

The proposed method of estimating the local direction and equivalent length of the line interval represented by a given line element is local in the sense that estimation is done on the basis of only the directions of elements immediately neighbouring this element. In the case of a square grid, there are six neighbouring elements (Fig. 3 a), three at each end of the central one. Each of these six elements can be present or absent - on or off. The central line element – horizontal or vertical – together with its immediate neighbours (those present – 'on') will be referred to as a *line fragment*. A line element which is central in a given line fragment is a neighbour element in the subsequent line fragment, if there is one. What is seen in Fig. 3 a can be considered as the first-order

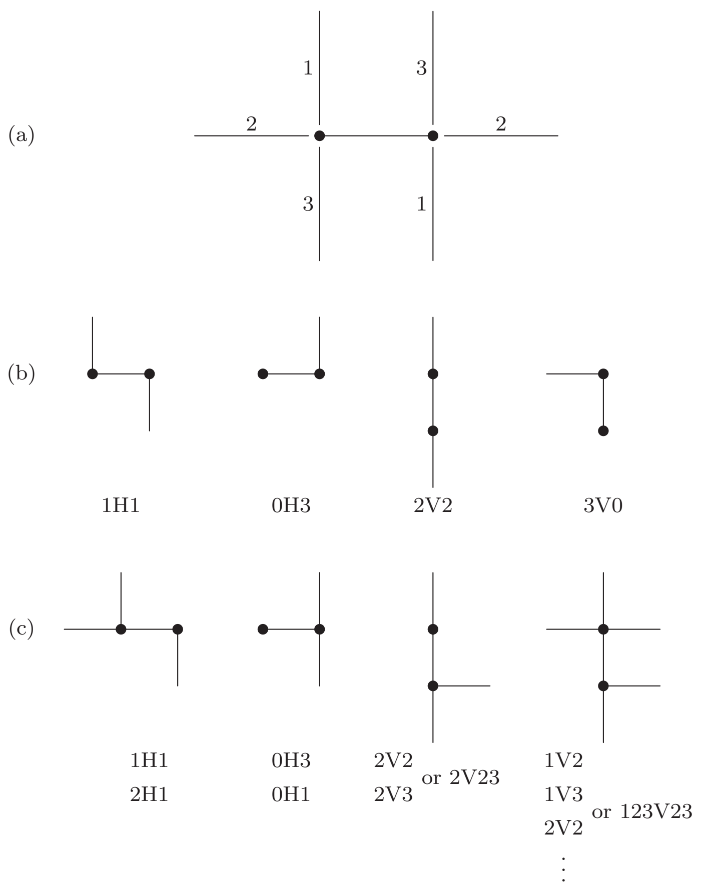


Fig. 3. Line fragments: (a) general layout: a central line element and its six immediate neighbours, three at each end; central element can be horizontal: H or vertical: V; directions of the neighbouring elements are denoted with numbers 1–3, and 0 denotes absence of a neighbouring element at an end; (b) examples of line fragments and their codes – without and (c) – with forks; forks make multiple codes possible. Fragments (b) 0H3, 3V0, and (c) 0H3 and 0H1 represent line ends.

neighbourhood of the central element. Higher order neighbourhoods can also be defined. In this paper we shall restrict our considerations to the first-order one.

In Fig. 3 b and c some examples of line fragments are shown. Each fragment can be represented with the coded position of its central element and relative positions of its neighbours. Fragments with horizontal and vertical central elements will be called horizontal and vertical, respectively. They differ only in the inclination angle, by 90°; for now, we shall consider only horizontal fragments for clarity. Line fragments with forks (Fig. 3 c) can be treated as multiple line fragments — they can be described with multiple codes. Treatment of such fragments depends on the requirements of a specific algorithm in which local estimated line length and direction is used. An example will be given in Chapter 4.

In further text such fragments as e.g. (1H1, 1V1 and 3H3), or (3H2, 1H2 and 2H1), will be referred to as *similar* and will be discussed together, as they can be transformed into each other with isometric transformations.

The method of estimating the line direction and length is based on a single assumption concerned with its locality. The locality means that a single line fragment is available at a time. It is assumed that the layout of the current line fragment is representative for the shape of the line it is the part of.

The values of the local equivalent direction and length, which will be derived now, refer only to the central element of each line fragment. The line elements belonging to the neighbourhood of the central element will be assigned their own equivalent values, when they are considered as central elements of subsequent line fragments.

Let us take a line fragment denoted 3H3 (Fig. 4 a) and build a stepped curve of this fragment as its element. We shall obtain the line similar to that from Fig. 2 b. The line fragment 3H2 will give rise to a stepped curve as in Fig. 4 b.

The line fragment 3H3 (or 1V1) projects itself onto an interval of the equivalent straight line of length $\sqrt{2}/2$, and will have equivalent length $\sqrt{2}/2 \approx 0.707$, and equivalent direction north-east (45°). Line fragments 1H1, 3V3 will have the same length, and the direction will be north-west.

There are two possible ways of calculating the equivalent length of the line fragment 3H2 (cf. Fig. 4 b). The first is the following: three line elements give a straight line interval of length $\sqrt{5}$, so the equivalent length of each of them is $\sqrt{5}/3 \approx 0.745$. The second possibility is that a straight line interval of length $\sqrt{5}$ is formed by a single element α of type 1V1 with equivalent length $\sqrt{2}/2$, and two elements: β of type 3H2 and γ of type 2H3, each with equivalent length x; therefore, $\sqrt{2}/2 + 2x = \sqrt{5}$ and $x = (\sqrt{5} - \sqrt{2}/2)/2 \approx 0.764$. The second result seems more conforming with the qualification of the line elements in the stepped curve, and the numerical experiments shown in Chapter 3 will confirm its better accuracy (see Table 3, Figure 5). The equivalent direction of line fragments of the type similar to 3H2 is inclined with respect to the horizontal or vertical direction by $\pm \arctan(1/2) \approx \pm 26.6°$, depending on its specific location.

Assigning the equivalent values to the line fragments of type 2H2 or 2V2 (see Fig.3), 1H3 or 3V1, 0H1, or one-element fragments 0H0 and 0V0 is straightforward and needs

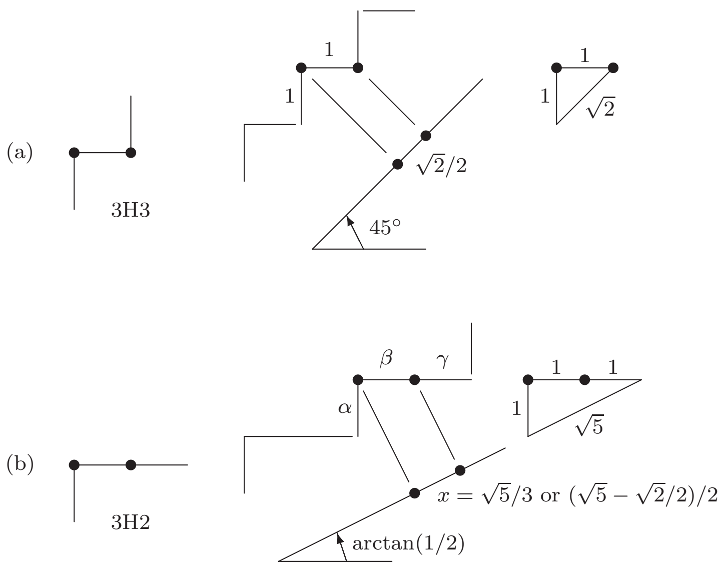


Fig. 4. Calculation of equivalent length and direction for line fragments (see text); (a) fragment of type 1H1 – and 3H3, 1V1, 3V3; (b) fragment of type 3H2 – and the similar – equivalent length can be calculated in two ways (see text); the fragment is formed of a single line element α of type 1V1 and two elements β and γ of type 3H2 and 2H3, respectively, both of the sought equivalent length x.

no explanation.

The values of the equivalent lengths and directions for the sets of similar line fragment types is summarized in Table 1. The set of similar line fragments listed in this table under a separate reference number can be denoted as a *class*.

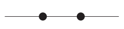
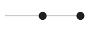
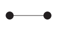
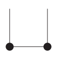
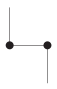
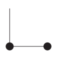
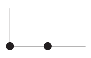
A simple way of representing the equivalent lengths or directions for line fragments is a look-up table. The 2D, 4×4 LUT is given in Table 2.

The global length of a curve represented by a string of line elements can be calculated by assigning its elements to the classes 1–7, according to the shape of the respective line fragment for each element, and summing their equivalent lengths.

3. Simple Numerical Examples

The straightforward concept presented in the previous chapter has been verified first on the series of examples of simple geometrical shapes – squares and circles – for which the stepped curve representation as well as the ideal geometry is fairly obvious. These shapes have been displayed in Fig. 5.

The value by which the shapes will be compared is their perimeter. The accurate values of the perimeter for the ideal curves will be denoted p. The values obtained

class	types	shape	equiv.length	equiv.angle
1	2H2, 2V2		1	0
2	2H0, 0H2, 2V0, 0V2		1	0
3	0H0, 0V0		1	0
4	1H3, 3H1, 1V3, 3V1		1	0
5	1H1, 3H3, 1V1, 3V3		$\sqrt{2}/2$	45°
6	1H0, 0H1, 3H0, 0H3, 1V0, 0V1, 3V0, 0V3		$\sqrt{2}/2$	45°
7	1H2, 2H1, 3H2, 2H3, 1V2, 2V1, 3V2, 2V3		$(\sqrt{5}-\sqrt{2}/2)/2$	arctan(1/2)

Tab. 1. Values of equivalent lengths and directions for the classes of similar line fragment types. The equivalent angle is measured with respect to the central element of the fragment; the sign of the angle varies with the line fragment type within a class and is not specified here.

index	0	1	2	3
0	—	0.707/ −45.0	1.000/ 00.0	0.707/ +45.0
1	*sym.*	0.707/ −45.0	0.764/ −26.6	1.000/ 00.0
2	*sym.*	*sym.*	1.000/ 00.0	0.764/ +26.6
3	*sym.*	*sym.*	*sym.*	0.707/ +45.0

Tab. 2. Look-up table of the values of equivalent lengths[pixels]/directions[deg] for line fragment types. Indices 0–3 represent neighbouring elements directions, according to Fig. 3: row index represents left (H fragment) or upper (V fragment) element, column index – right (H fragment) or lower (V fragment) element. Equivalent direction is represented with the angle by which the central element of the fragment should be turned anti-clockwise. The given approximate numerical values represent: $0.707 \approx \sqrt{2}/2$, $0.764 \approx (\sqrt{5}-\sqrt{2}/2)/2$ and $26.6° \approx \arctan(1/2)$; the remaining values are accurate.

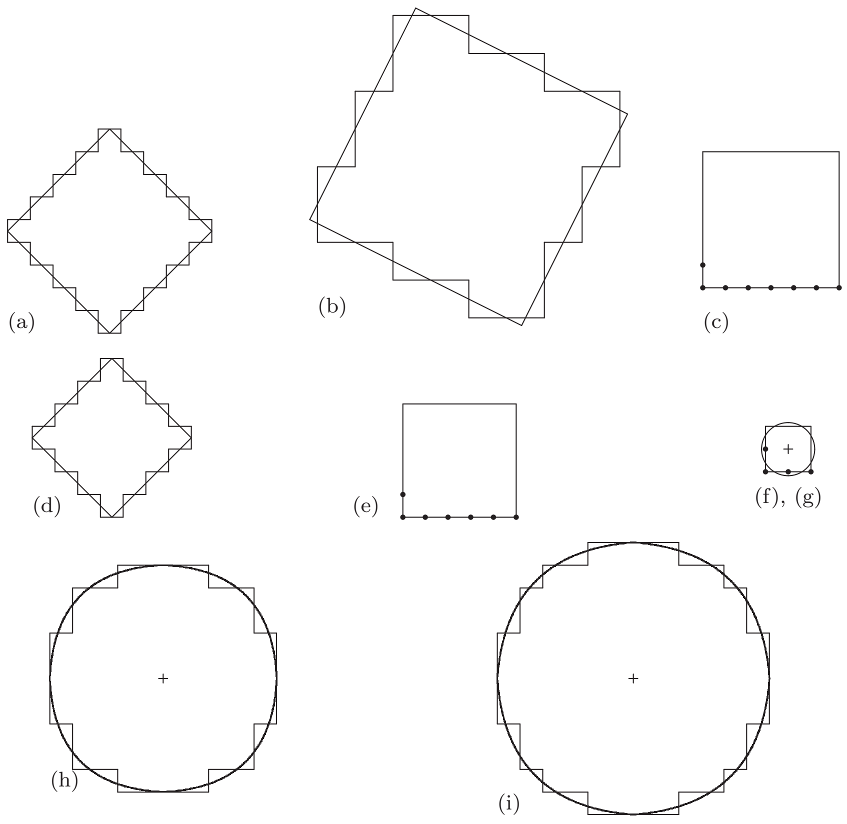


Fig. 5. Shapes of the geometrical figures used in the examples and their stepped curve representations (see text for details): (a) square, edge ∼6, inclination angle 45°; (b) square, edge ∼6, angle ∼27°; (c) square, edge 6, not inclined; (d) square, edge ∼5, angle 45°; (e) square, edge 5, not inclined; (f) square, edge 2, not inclined; (g) circle, radius ∼1; (h) circle, radius ∼5; (i) circle, radius ∼6

fig.	shape	a or r	p	$p_{\sqsubset}$	$p_{\subset}$	$\delta(p_{\sqsubset})$ [%]	$\delta(p_{\subset})$ [%]
a	$\square 6,\ 45°$	6.40	25.6	36	26.6 (26.6)	+41	−4.0 (−4.0)
b	$\square 6,\ 27°$	6.32	25.3	28	24.0 (23.6)	+26	−5.1 (−6.9)
c	$\square 6,\ 0°$	6.00	24.0	24	22.1 (21.7)	0.0	−7.9 (−9.8)
d	$\square 5,\ 45°$	5.00	20.0	28	21.0 (21.0)	+40	−4.9 (−4.9)
e	$\square 5,\ 0°$	5.00	20.0	20	18.1 (18.0)	0.0	−9.4 (−10)
f	$\square 2,\ 0°$	2.00	8.00	8	6.12 (5.96)	0.0	−24 (−25)
g	$\bigcirc 1$	1.13	7.09	8	6.12 (5.96)	+13	−14 (−16)
h	$\bigcirc 5$	5.04	31.7	40	32.0 (31.5)	+26	+0.9 (−0.5)
i	$\bigcirc 6$	5.97	37.5	48	37.7 (37.2)	+28	+0.4 (−0.8)

Tab. 3. Comparison of the accurate and approximate perimeters for the geometrical shapes from Fig. 5 (a–i); $\delta(x)$ is the relative error of x with respect to p. For line fragments of class 7 (e.g. of type 3H2, cf. Table 1) the equivalent length was $(\sqrt{5}-\sqrt{2}/2)/2 \approx 0.764$ (cf. Fig. 4 with explanations); for comparison, the values obtained for that length equal to $\sqrt{5}/3 \approx 0.745$ are given in the ellipses $(\cdot)$.

by simply counting the line elements will be called the *counted* values, and will be denoted $p_{\sqsubset}$. The values obtained with the use of the derived equivalent lengths of the line fragments will be called the *equivalent* values, and will be denoted $p_{\subset}$.

Relation between the geometries of the ideal figures and their stepped representations has been established with the assumption that the area A of the ideal figure and that of the stepped one are equal. Thus, the perimeters of the ideal figures p to be compared with the equivalent ones $p_{\subset}$ and the counted ones $p_{\sqsubset}$, can be calculated for squares as

$$p^{\square} = 4\sqrt{A} \tag{1}$$

and for circles as

$$p^{\bigcirc} = 2\pi\sqrt{A/\pi} \tag{2}$$

Having the perimeters, the lengths of the edges of ideal squares a and radii of ideal circles r can be easily found.

The accurate and approximate perimeters for the figures have been compared in Table 3.

The following remarks can be made on this simple comparison:

- The equivalent values $p_{\subset}$ are more accurate than the counted ones $p_{\sqsubset}$ in all the cases, except those for which $p_{\sqsubset}$ is accurate, which takes place only for rectangles with horizontal/vertical edges.
- The larger the share of inclined (cf. cases a,b,c) or curved (cf. cases g,h,i) lines in the ideal contour, the bigger the advantage of the equivalent values $p_{\subset}$ over the counted ones $p_{\sqsubset}$.
- The smaller the figure the worse the accuracy of the equivalent values $p_{\subset}$. This results from that for small figures the share of line elements which badly represent the local

curve shape is larger. Case (f) is the worst one: none of the line fragments of class 7 (2H1 etc.) represent right angle vertices; nevertheless, the relative error for this worst case is -24%, which is much better than the maximum relative error for the counted values, which equals $+41\%$.
- The equivalent values obtained with the equivalent length for the line fragments of class 7 (2H3 etc.) equal to $(\sqrt{5}-\sqrt{2}/2)/2 \approx 0.764$ are in general more accurate than those obtained with that value equal to $\sqrt{5}/3 \approx 0.745$. The first value should be used for fragments of class 7.

The general conclusion from the presented considerations is that, in spite of the simplicity of the method, the use of the equivalent lengths for line elements in the calculation of curve lengths is quite advantageous.

4. Application to the 'Weak Membrane' Image Restoration Algorithm

The 'weak membrane' image restoration method has been described in detail in [11]. The overview given here covers only the topics necessary from the point of view of our curve direction and length estimation algorithm[1].

The method realizes two tasks, simultaneously:

- elimination of high-frequency noise, i.e. filtration;
- detection of edges, defined as sets (or strings) of inter-pixel boundaries in which large intensity differences between neighbouring pixels occur.

The tasks are realized by minimizing a function E, defined on the input image $d(x,y)$ and the output image $u(x,y)$

$$E = \int_S (u(x,y)-d(x,y))^2\, dS + \lambda^2 \int_S grad(u(x,y))\, dS + \int_L \alpha\, dL \tag{3}$$

where $x, y \in S$, S is the area of the input as well as the output image, and L is the total length of the edges in the image. The function E can be considered as the energy of the elastic membrane, with its shape (out-of-plane deflection) described by $u(x,y)$. The membrane is attached to an elastic base, the initial shape of which is described by $d(x,y)$. λ represents the ratio of stiffnesses of the membrane and the base. The membrane can break, and the energy of the break of unit length is proportional to α.

The parameters λ and α act in the following way. λ is the scale of the filter; if the membrane were compared to the Gaussian filter, λ could be treated as analogous to 3σ – the 'influence range' of the filter. In other words, the larger λ is, the more the high-frequency component in the image is attenuated. With α one can adjust the sensitivity of the edge detector: a straight, isolated edge will be detected if its height is not less than

$$h_0 = \sqrt{2\alpha/\lambda} \tag{4}$$

The threshold height h_0 is valid only for edges which are far from each other with respect to the scale λ; the nearer two edges are, the larger the real threshold is. Also, steep slopes

[1]The non-invariant version of the algorithm with fixed edge sensitivity threshold has been applied.

may give rise to edges if the intensity gradient is larger than

$$h_1 = \sqrt{\alpha/2\lambda^3} \tag{5}$$

Double edge (two near, parallel edges on a single slope) occurs if the gradient is larger than

$$h_2 = 2\sqrt{\alpha} \tag{6}$$

In practice, λ and h_0 are considered as the algorithm parameters. λ should be large enough to provide for sufficient filtration, and not too large in order to avoid false edges formation (h_1). False edges can be avoided if α (and h_0) is large; however, if one wants to let the membrane break, h_0 should be kept small enough.

The algorithm for finding $u(x, y)$ satisfying Eq. 3 is an iterative algorithm which operates on a square mesh of image pixels. What will be important for us is that in the subsequent iterations the edge elements (the inter-pixel boundaries) are classified as broken, unbroken, or ambiguous ('yet unknown'). While the algorithm proceeds, the number of ambiguous edge elements (*edgels*) decays, and when the minimum of E is reached, the state of all the edges is known. For the detailed presentation of the algorithm, and the discussion of the conditions for edge detection, please refer to [11]. That algorithm will be referred to as the *basic algorithm*.

Now let us briefly list the advantages and disadvantages of the 'weak membrane' filter. The following advantages in comparison to the conventional edge-finding and filtering algorithms based on blurring and maximum gradient detection (as e.g. the Laplacian-of-Gauss) can be listed:

1. Edges are found during filtration; this non-linear optimization process yields excellent edge stability in the scale-space.
2. Edges in the image are very well preserved; also their joints are not disconnected.
3. The above holds also in noisy images.
4. The entire area of the input image is processed — no half-scale wide stripes along the image boundary are lost.
5. The algorithm copes with sparse data in a natural way: the first term in Eq. 3 is omitted for pixels where no data is provided.

The algorithm has some disadvantages, too. They are as follows:

1. The optimization algorithm based on Eq. 3 tends to produce uniform brightness regions separated with edges[2].
2. Edge sensitivity threshold depends on the mutual layout of the edges.
3. False edges are found if the slope of the brightness function is large[3].
4. The algorithm is a relatively time-consuming one (cf. Sections 5 and 6).
5. Sensitivity to inclined edges is smaller than that to horizontal or vertical edges.

[2]the 'weak plate' is better in this respect, but much less efficient [11]

[3]the 'invariant membrane', which, in turn, reduces the gradients to greater extent, or the 'weak plate' can help alleviate this problem

What will interest us is the way of overcoming the disadvantage 5 of the 'weak membrane' algorithm. Its source is the difference between the real length of an edge and its estimated length, replaced in the basic algorithm by the number of line elements in which the edges occur (elements where the membrane has broken), what has been illustrated in Fig. 2.

It can be said that the basic algorithm models the membrane with non-isotropic sensitivity to edges. The algorithm proposed below will represent the membrane with isotropic sensitivity; we shall call that algorithm the *modified* or the *isotropic* algorithm. We shall also use the notion of the *'isotropic membrane'*, although the membrane itself, as expressed by the Equation 3, is always isotropic — it is only the mesh effect in the numerical algorithm that makes it anisotropic.

In the case of Fig. 2, the real edge length is $3\sqrt{2}$, and its energy should be $3\sqrt{2}\alpha$. The basic algorithm, however, would ascribe it the length 6, and the energy would be 6α, which is $\sqrt{2} \approx 1.41$ times too much. Because the shape of the pixel mesh is fixed, we cannot change the number of the counted edgels; what we can do, is adjust the value of α, according to the equivalent length of the edge element, for which the energy of the break is calculated. In our modified algorithm, α will be replaced by α_e. To compensate for the energy error in the above example of an edge inclined by 45°, α_e should be set to $\alpha/\sqrt{2}$. If we look at Eq. 4 we can see that this replacement will reduce the threshold h_0 by $\sqrt[4]{2}$, i.e. $h_{0e} = h_0/\sqrt[4]{2} \approx h_0/1.19$. This means that for such edges the non-modified threshold is about 19% too high.

According to the Tables 1 and 2, in the modified algorithm there are three types of the line elements, differing only in their behaviour in the Eq. 3, that is, having different equivalent lengths. As these elements are inclined with respect to the horizontal (or vertical) direction by an angle 0°, ≈27° or ≈45°, let us denote them as the class $CLA00$ elements, $CLA27$ and $CLA45$ elements, respectively. The class $CLA00$ contains the edge elements of classes 1, 2, 3 and 4 from Table 1, the class $CLA27$ — the elements of classes 5 and 6, and the class $CLA45$ — the elements of class 7. The equivalent values of α_e, which now replaces α in the algorithm, result from the values of the equivalent lengths of the line elements from Tables 1 and 2. For the elements from $CLA00$ it is $\alpha_e = 1.000$, of $CLA27$ — $\alpha_e \approx 0.764$, and of $CLA45$ — $\alpha_e \approx 0.707$.

The classes $CLA00$, $CLA27$ and $CLA45$ are re-considered and assigned just before α_e is used, i.e. before each decision on the existence or non-existence of a break is made, so that always the most up-to-date information is used.

Initially, the lowest possible value of $\alpha_e = 0.707$ is set ($CLA45$) to let the edges appear more freely; as the iterations proceed and the information on the edges becomes more reliable, α_e is adjusted according to that information, and the surplus edges disappear. This should also enhance the edge continuity and the joints-preserving ability of the basic algorithm.

In the iteration process, α_e is adjusted for each line fragment by increasing it "as slowly as possible". First, the smallest possible α_e for the current state of the line elements of the considered line fragment is found. Then, as a new value of α_e, the larger one of the two values is set: either that smallest value, or the previous value of α_e in the considered element. To find the smallest possible α_e all the possible layouts are checked for the fragment. Multiple possibilities emerge if in the fragment there are forks or edgels in an ambiguous state: always that possible layout which yields minimum α_e is assumed. If there is only one possibility, the decision is trivial. Finally, if the central element is in the 'not broken' state (no edge), the minimum value $\alpha_e = 0.707$ is taken (as for $CLA45$).

It is important that α_e never be decreased. Otherwise, the following oscillations of edges could appear. Let the initial α_e be 0.707 ($CLA45$), and assume that an edge of $CLA00$ is found. Therefore, α_e is set to 1.000. Let us now assume that the edge is discarded with that α_e. If α_e is then reduced back to 0.707, the edge can re-appear in the subsequent iteration. This yields the increase of α_e back to 1.000, and *ab initio*.

The 'isotropic membrane' algorithm gives the expectedly better results, at the expense of some extra operations. In the examples presented in the Section 5 the numbers of iterations and the times of calculation are given for comparison. The additional memory requirements are not very high. For storing the edge state information (break, no break, ambiguous) two bits are enough; the same holds for the edge class information ($CLA00$, $CLA27$, $CLA45$). For each pixel there are two inter-pixel boundaries (except the picture side pixels); it results that eight additional bits per pixel must be stored.

5. Experiments with the Modified 'Weak Membrane' Algorithm

The presentation of the results obtained with the basic and the modified algorithm for artificial images shown in Figs 6 – 9 is now in order. The $256 * 256$, 256-grey-level images were designed to amplify the effect of the edge sensitivity dependence on the edge direction. Grey levels of the dark and bright regions change from 172(bright)/82(dark) at the circle centre (row 185, column 185) to 192/62 at the ends of the horizontal/vertical circle radii. The spacing of radii is 15°. It was attempted to obtain 'nearly quadrangular' blobs in the image from Fig. 6; the distances between circles are 20 pixels. In Fig. 7 the circles are twice denser, to check if the mutual influences between edges will interfere with the effect of the modification of the algorithm. The Figs 8 and 9 show the versions of the test images with the noise added (centred Gaussian noise, $\sigma = 8$). Before adding the noise to obtain Fig. 9 from Fig. 7, the image from Fig. 7 was blurred with a $3 * 3$ Gaussian filter. In this way, the effect of blurring on the performance of the modified membrane filter can be seen.

The results of the basic algorithm as well as the modified one are shown in the Figs 10 – 17, and the cross-reference between the original and the filtered images, the membrane parameters and the numbers of iterations and calculation times are all collected in the Table 4.

All the results have been obtained with the same values of λ and h_0, except those for the images blurred with the Gaussian filter, where the smaller contrast necessitated for the smaller threshold h_0.

Note that the more edges there are in the image, the larger is the calculation overhead for edgel classification in the isotropic algorithm, with respect to the basic one.

#	input Fig.	output Fig.	version	iterations	time[min]
1	6 (coarse, pure)	10	B	83	11
2	6 (coarse, pure)	11	I	90	16
3	8 (coarse, noisy)	12	B	88	13
4	8 (coarse, noisy)	13	I	95	21
5	7 (dense, pure)	14	B	88	12
6	7 (dense, pure)	15	I	91	17
7	9 (dense, noisy)	16	B	112	16
8	9 (dense, noisy)	17	I	118	25

Tab. 4. Comparison of the results obtained with the basic (B) and isotropic (I) version of the 'weak membrane' algorithms. In all the cases $\lambda = 2.0$, $h_0 = 105.0$ except ## 7 and 8, where $h_0 = 74.0$. Times of calculation with a 80486DX, 50MHz AT computer; the programming language was Borland C++ v. 3.0.

In all the figures, the edges found by the weak membrane have been marked with white pixels (a pixel is white if an edge occurred in the inter-pixel boundary below or to the right of that pixel).

It can be clearly seen In the Figs 10, 12, 14 and 16 that the edge sensitivity of the basic algorithm depends heavily on the edge direction.

Let us analyse the Fig. 10. The first vertical edge on the circle (marked as a in the Figure) is found on the fifth circle. The brightness levels of the original image are 182/72 there and their difference is 110, which is near to the value of $h_0 = 105$ used in the algorithm. The first edge inclined at 45° appeared on the eighth circle (b in the Fig.), where the original brightness levels were 189/65, difference 124. This difference is ≈18% larger than the threshold h_0, which is in good conformity with the relative error of edge sensitivity for such edges, calculated in the previous Section on the basis of the edge energy estimation error.

Now, let us repeat the same analysis for the edges on the radii. The first horizontal edge on the horizontal radius (c in the Fig.) originated between the fifth and the sixth circle, with the brightness difference about $184 - 70 = 114$, and on the skewed radius (d) — between the ninth and the tenth circle, difference $192 - 62 = 130$. The edge sensitivity errors with respect to $h_0 = 105$ are: horizontal edge — 8.5%, skewed edge — 24%. The skewed edge appeared at the brightness difference 14% higher than that at which the horizontal one was found. These results are similar to those obtained for the edges on the circles.

In the Figs 11, 13, 15 and 17, where the results obtained with the 'isotropic membrane' are shown, the dependence of the edge sensitivity on the edge direction nearly disappears (a weak effect adverse to that observed for the basic algorithm can be noticed). Also the edge joints are much better preserved.

The positive effect of adjusting the edge sensitivity is similar in all the images, independent of the presence or absence of noise or blur.

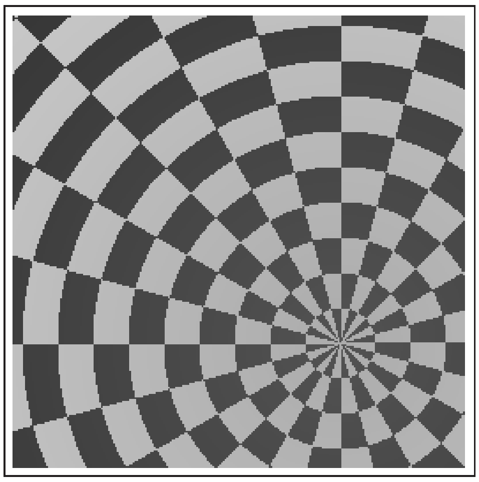


Fig. 6. The test image with coarse mesh, without noise.

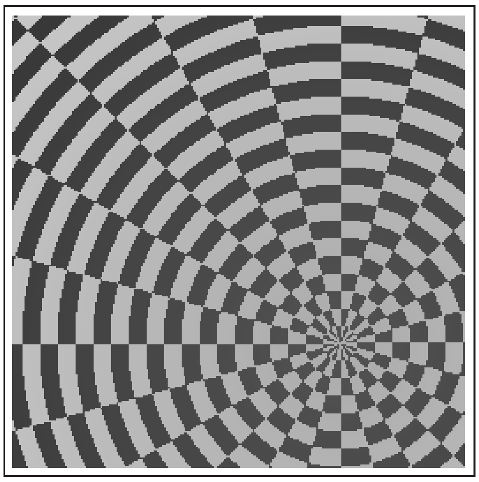


Fig. 7. The test image with dense mesh, without noise.

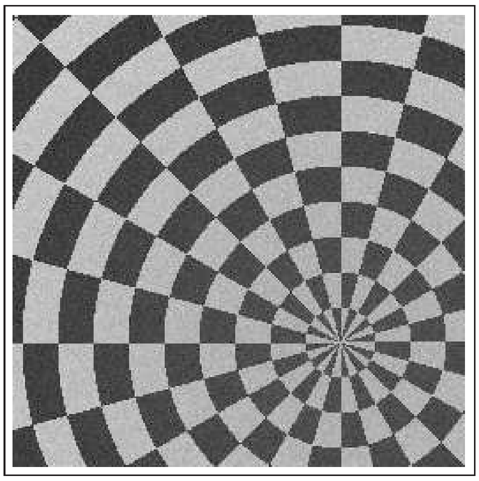


Fig. 8. The test image with coarse mesh, with noise.

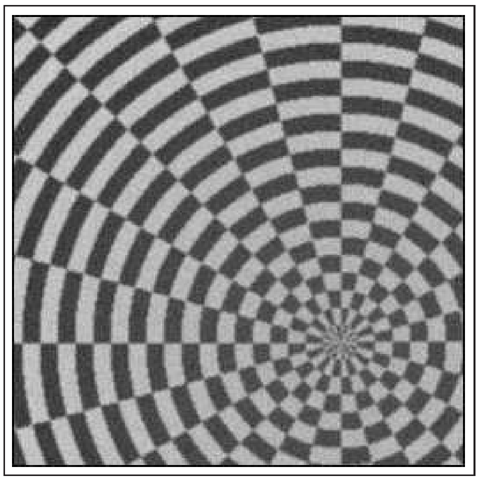


Fig. 9. The test image with dense mesh, blurred, and with noise.

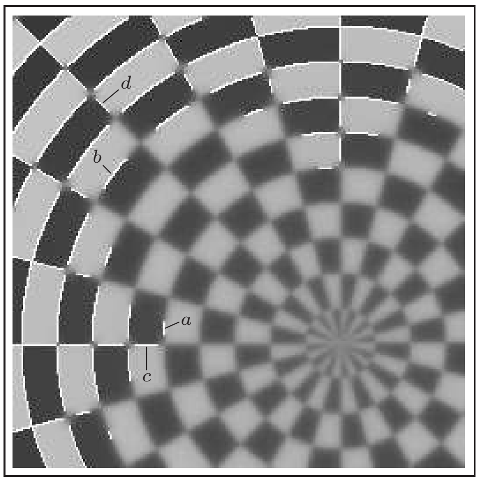


Fig. 10. The image of Fig. 6 (coarse mesh, no noise) processed with the basic algorithm; a ÷ d: see text.

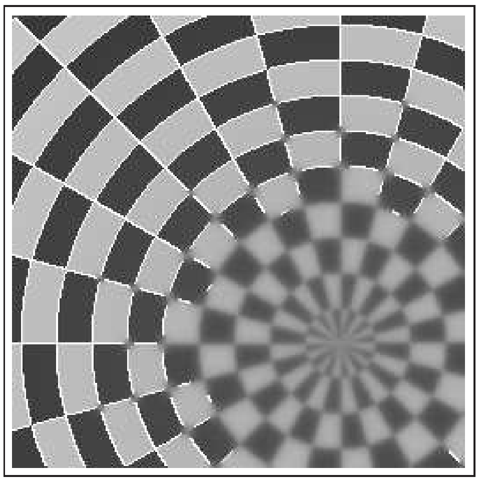


Fig. 11. The image of Fig. 6 (coarse mesh, no noise) processed with the isotropic algorithm.

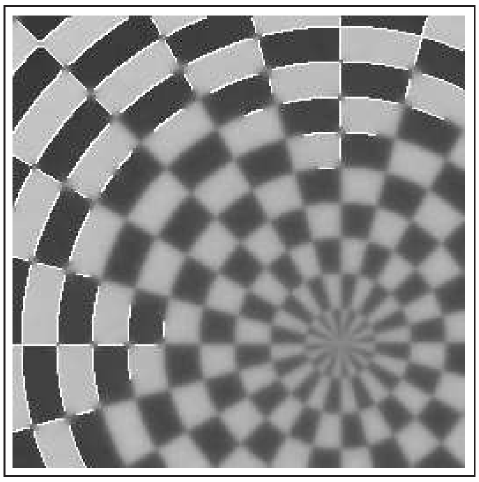


Fig. 12. The image of Fig. 8 (coarse mesh, with noise) processed with the basic algorithm.

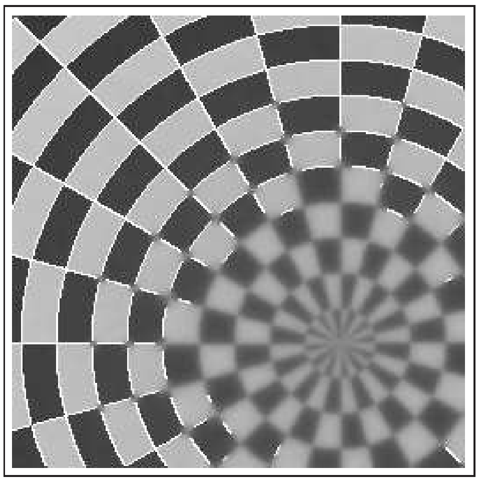


Fig. 13. The image of Fig. 8 (coarse mesh, with noise) processed with the isotropic algorithm.

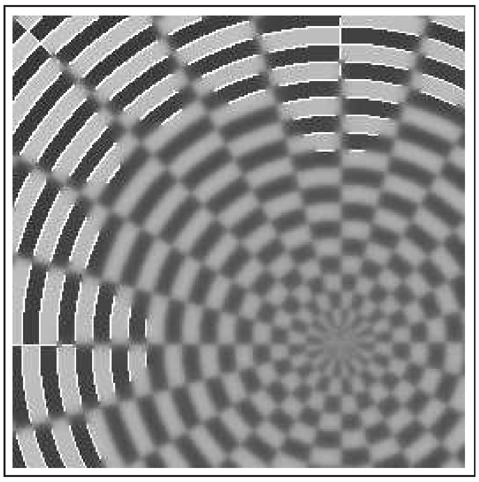


Fig. 14. The image of Fig. 7 (dense mesh, no noise) processed with the basic algorithm.

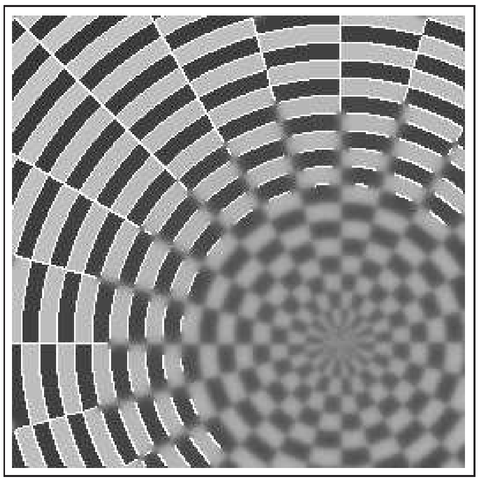


Fig. 15. The image of Fig. 7 (dense mesh, no noise) processed with the isotropic algorithm.

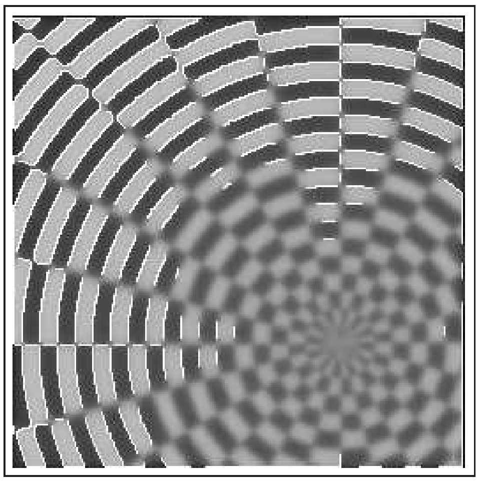


Fig. 16. The image of Fig. 9 (dense mesh, with blur and noise) processed with the basic algorithm.

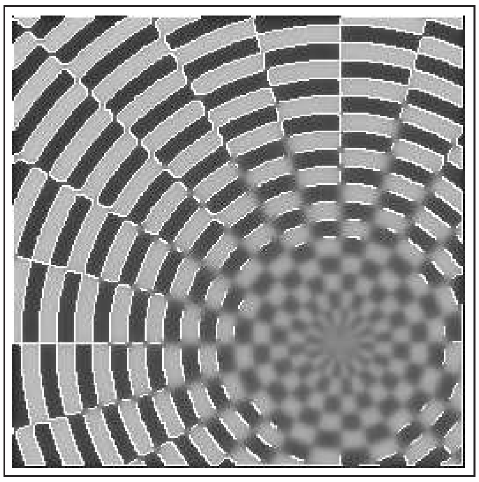


Fig. 17. The image of Fig. 9 (dense mesh, with blur and noise) processed with the isotropic algorithm.

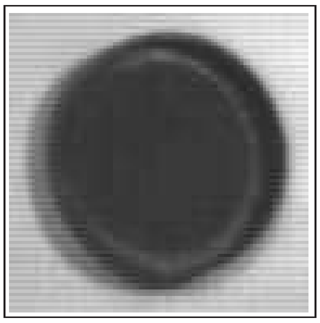


Fig. 18. The original image for the comparison of filters.

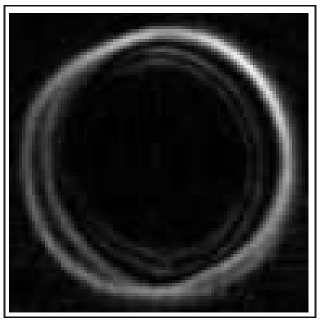


Fig. 19. Result of Gaussian filtering and then Sobel edge finding.

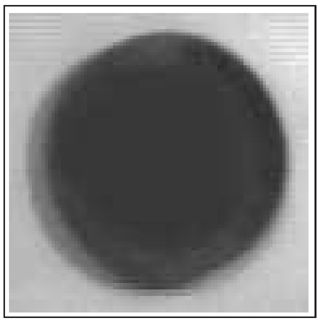


Fig. 20. Result of adaptive median filtering.

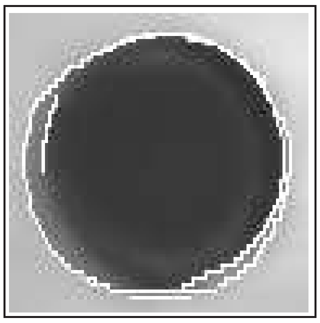


Fig. 21. Result of 'membrane' filtering — basic algorithm.

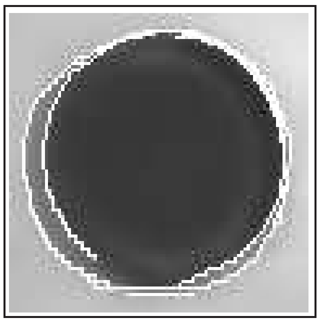


Fig. 22. Result of 'membrane' filtering — isotropic algorithm.

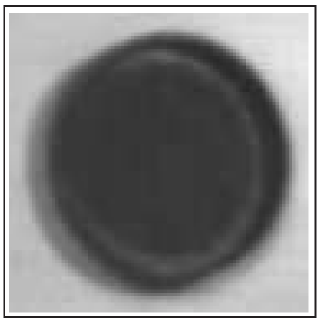


Fig. 23. Result of adaptive Fourier filtering.

6. Comparison of the filtering and edge finding methods

In the previous Section, the basic and the isotropic versions of the 'weak membrane' image restoration algorithm were discussed on the basis of the results obtained for artificial images. Now, let us have a look at the outputs of a handful of some known methods which produce the results comparable to those of the membrane. This can show us, what the membrane can, and what it cannot do. The original image used for the comparison is a poor-quality $100 * 100$, 256-grey-level image of a camera lens cap, with some noise and shades (Fig. 18). The filters used are described below, in the ascending order of the times of calculation (80486DX, 50MHz AT computer; programming language: Borland C++ v. 3.0).

Gaussian filter and Sobel edge finder The original image was filtered with a $3 * 3$ Gaussian filter, and then the edges were found with a Sobel operator. The image in

Fig. 19 is a non-thresholded edge image.
Calculation time (Gauss+Sobel): $0.18s + 0.55s = 0.73s$.

Adaptive median filter The result after three passes is shown in Fig. 20. Each pixel of the output is a result of a 1D median filter applied to a $1 * 9$ window of the input, located in one of the eight directions (N-W, NNE-SSW, NE-SW, ...). The nearest direction to the normal to the gradient is chosen. The gradient of a local least-mean-square plane approximation of the brightness function in a $9 * 9$ window is used [16].
Calculation time: $9s$.

Membrane – the basic algorithm The parameters were: $\lambda = 2$, $h_0 = 34$. The result obtained in 45 iterations is shown in Fig. 21[4].
Calculation time: $60s$.

Membrane – the isotropic algorithm In the Fig. 22 it can be noticed that some inclined edges, not found by the basic algorithm, have been found now. In some regions edge smoothness is also a little better. The parameters were the same as for the basic algorithm, and the number of iterations was 41.
Calculation time: $68s$.

Adaptive Fourier filter A directional Fourier filter [16] was applied, with the direction found with the method from [12] (Fig. 23). All the frequencies except the first three lowest ones were deleted. The window dimensions for direction finding was $9 * 9$, and the window in which the local FFT filter was applied in that direction was $16 * 16$.
Calculation time: $405s$.

7. Conclusion

A simple method for estimating local length and direction of a curve represented as a set of inter-pixel boundaries has been proposed. The method consists in classifying the line elements as belonging to one or more of a small number of classes, depending on the positions of the immediately neighbouring elements. The properties of the classes can be stored in a small look-up table. No "smooth" representation in a form of a polynomial or a spline fitted to the edge elements is used.

Simple numerical examples with closed curves (contours of geometrical figures) indicate that the accuracy of curve length estimation is not worse than 10% for smaller and 1% for larger figures.

The method has been applied to enhance the 'weak membrane' image restoration algorithm [11]. The original numerical implementation of that algorithm suffers from the dependence of sensitivity to edges on the edge direction (mesh-induced anisotropy of the edge sensitivity). The local direction estimation method made it possible to make the algorithm isotropic. As a side-effect, the edge continuity-preserving capability of the method has been enhanced.

[4]Edge pixels are marked white, as in the Figs 10 – 13, Section 5

Acknowledgment

The research was financed by the Committee of Scientific Research (KBN) under the Grant No. 8 8055 91 02, realized in May 1992 – Apr 1993.

References

[1] Montanari U. : A note on minimal length polygonal approximation to a digitized contour, *Comm. ACM*, 13, 41-47.
[2] Kulpa Z. : Area and perimeter measurement of blobs in discrete binary pictures, *CVGIP*, 6, 434-451.
[3] Ellis T.J., Proffitt D., Rosen D., Rutkowski W. : Measurement of the length of digized curved lines, *CVGIP*, 10, 333-347.
[4] Proffitt D., Rosen D. : Metrication errors and coding efficiency of chain-encoding schemes for the representation of lines and edges, *CVGIP*, 10, 318-332.
[5] Pavlidis T. : *Grafika i Przetwarzanie Obrazów*, WNT, Warsaw 1987. (*Algorithms for Graphics and Image Processing*, Computer Science Press Inc., Rockville, 1982.)
[6] Kulpa Z. : More about areas and perimeters of quantized objects, *CVGIP*, 22, 268-276.
[7] Pavlidis T. : Curve fitting with conic splines, *ACM Trans. Graphics*, 2(1), 1-31.
[8] Geman S., Geman D. : Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images, *IEEE Trans. PAMI*, 6(6), 721-741.
[9] Nalwa V.S., Pauchon E. : Edgel aggregation and edge description, Proc. 8th Int.Conf.Pattern Recognition, Paris, 27-31 Oct, 1986, 604-609.
[10] Wall K. : Curve fitting based on polygonal approximation, Proc. 8th Int.Conf.Pattern Recognition, Paris, 27-31 Oct, 1986, 1273-1275.
[11] Blake A., Zisserman A. : *Visual Reconstruction*, MIT Press.
[12] Kass M., Witkin A. : Analyzing Oriented Patterns, *CVGIP*, 37, 362-85.
[13] Chalmond B. : Image Restoration Using an Estimated Markov Model, *Signal Processing*, 15(2), 115-129.
[14] Zucker S.W., David Ch., Iverson L. : The organization of curve detection: Coarse tangent fields and fine spline covering, Proc. Workshop: From Pixels To Features, Bonas, France, 22-27 Aug, 1988, 75-89.
[15] Dolan J., Weiss R. : Perceptual grouping of curved lines, Proc. DARPA Image Understanding Workshop, Palo Alto, CA, 23-26 May, 1989, 1135-1145.
[16] Chmielewski L., Skłodowski M., Cudny W., Nieniewski M., Kuriański A., Michalski B. : Fringe image processing in the White Light Wavelength Stepping Method, Report of the Grant KBN 8 8055 91 02, EPSILON ARG, Warsaw, Apr 1993.