Czech Pattern Recognition Workshop 2000, Tomáš Svoboda (Ed.)
Peršlák, Czech Republic, February 2–4, 2000
Czech Pattern Recognition Society

Corners toolbox allowing processing binary images in a compressed form*

Luboš Bureš, Karel Hanton, Csaba Meszaros, Jan Paleček, Tomáš Telenský
Charles University, Faculty for Mathematics and Physics, Computer Science Section
118 00 Praha 1, Malostranské náměstí 25, Czech Republic

Václav Hlaváč
Czech Technical University, Faculty of Electrical Engineering, Center for Machine Perception
121 35 Praha 2, Karlovo náměstí 13, Czech Republic
hlavac@cmp.felk.cvut.cz

Michail I. Schlesinger
International Research and Training Centre of Information Technologies and Systems
Ukrainian Academy of Sciences, 252207 Kiev, 40 Prospect Akademika Glushkova, Ukraine
schles@image.kiev.ua

Abstract *Traditionally, the time needed to process a binary image depends on the image size. Here, the image is losslessly compressed using Schlesinger's corner representation. The compressed image is represented by residuals to a* 2×2 *pixels predictor. There are significantly less such residuals than pixels of objects as residuals correspond to corners of object outlines only. The main benefit comes from the fact that set operations, translation and scaling can be performed on compressed images directly. The processing time depends on the number of corners (i.e. image complexity) and not on the number image pixels. The described public domain package implements operations AND, OR, XOR, NOT, scale, shift, connected component labeling and skeletonization.*

1 Introduction

This contribution describes the *toolbox Corners* alowing to compress losslessly binary image and process it in a compressed form. The toolbox is based on the theory proposed by M.I. Schlesinger [5] and his group. The theory is not widely known. The toolbox is a result of an ongoing software project, two semesters long, performed by five students. The teaching aim was to learn the team work during a software development.

The C_{++} package is unified by the user interface environment running under Microsoft Windows that allows to use individual modules. These are:

- Lossless compression/decompression of binary images. The compressed image is represented as a list of coordinates of predictor residuals (called corners).
- Basic set operations AND, OR, NOT, XOR, and scaling, shifting performing directly on compressed images.
- Connected component counting and labeling.
- Skeletonization of objects. The skeleton definition is slightly modified to perform in corner representation fast and in a way that is intuitively expected by humans.
- Beautification of the obtained skeleton, i.e. representing collection of shorter skeleton segments by longer ones. This skeleton generalization provides an effective vectorization of elongated objects in the image.
-

*This research was supported by the Czech Ministry of Education under the grant VS96049, the Research Topic JD MSM 212300013 Decision and Control for Industry.

2 Corner representation, basic theory

Let $T = \{(x, y) \mid 0 \leq x \leq m;\ 0 \leq y \leq n\}$ be a rectangular support in a plane. Let $v(x, y) \rightarrow \{0, 1\}$ be a binary image. The value 1 denotes (black in our images) pixels belonging to the objects, the value 0 indicates (white) background pixels. Let us suppose without loss of generality that left column and the bottom line of the image have value zero, i.e. $v(0, y) = v(x, 0) = 0$.

The basic entity used to represent the binary image v is a 2×2 window which we call probe in the sequel. The representative pixel $p_4 = v(x, y)$ of the probe is placed in the upper right corner of the probe, see Fig. 1. The other three pixels of the probe have the following meaning: $p_1 = v(x, y-1)$; $p_2 = v(x-1, y-1)$; $p_3 = v(x-1, y)$;

There are $2^4 = 16$ possible probe configurations that can be arranged into two rows and eight columns in such a way that probes in the same column differ by the bit in the representative pixel only, see Fig. 2. Eight out of sixteen possible configurations correspond to *corners* of objects in the image.

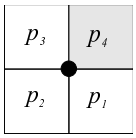


Figure 1: 2×2 probe.

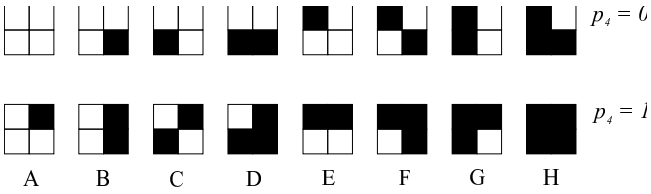


Figure 2: All 16 possible configurations arranged into eight pairs differing in the upper right bit only.

Four corners are *convex*, i.e. configurations A1, B0, C0, E0 and four corners are *concave*, i.e. configurations D1, F1, G1, H0.

Corners can be expressed easily mathematically. Let us define the *difference image* $d(x, y) = v(x, y) \oplus v(x, y - 1)$, where $\oplus$ denotes exclusive OR. The difference image indicates transitions from black to white and vice versa in horizontal direction. Transitions from black to white and vice versa in vertical direction in two neighboring lines of difference image d are given as $k(x, y) = d(x, y) \oplus d(x - 1, y)$. Corners K in the binary image v are given as $K(v) = \{x, y) \mid k(x, y) = 1\}$. The actual calculation needed to indicate corner in the 2×2 probe is extremely easy. The probe traversing the image v indicates corner if and only if it consists of odd black pixels, i.e. if and only if $k(x, y) = 1$,

$$k(x, y) = v(x, y) \oplus v(x-1, y) \oplus v(x, y-1) \oplus v(x-1, y-1).$$

In the *compression pass* leading to corner representation, the binary image v is traversed by the introduced 2×2 probe starting from the lower left position to the right and upwards. It is checked if the configuration is a corner or not in each probe position. In the former case, the value 1 (residual), and in the latter case the value 0 is written into corner represented image in the place of the currently placed representative point. There are significantly fewer corners compared to all object pixels for the typical binary image (e.g. the scanned engineering drawing). This yields much sparser corner represented image. The typical reduction factor is about 20 times!

The *decompression pass* starts again from the lower left corner of the compressed image moving to the right and upward. It is assumed that one column left to the image and below the image have value 0. All pixels in the probe are known with the exception of the representative pixel. The decompressed value can be uniquely determined due to paired probe configurations differing in the representative point only.

Region boundaries and corners lie between the pixels. That is the reason why each corner actually represents a point in the center of the probe that is denoted by a filled circle in Fig. 1. We shall use the same convention when showing corners pictorially.

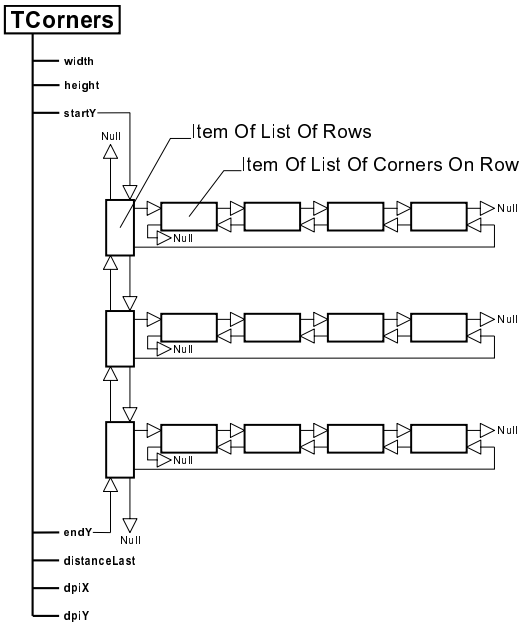


Figure 3: Data structure storing corners in our representation.

Images in corner representation are not implemented as matrix of corners. The list with pairs (x, y) corresponding to corners is used instead. The elements of the list are ordered by line coordinates. Corners in each line are ordered from left to right. Ordering allows to speed up operations with corner represented image. The actual structure is shown in Fig. 3.

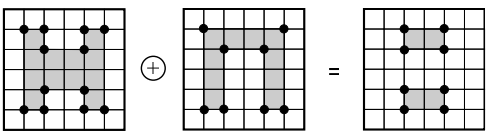
Let us note that the corner representation idea was developed further for the lossles compression of gray-scale images [2]. The probe properties were modified according to the image to achieve higher compression ratio at the expense of loosing the possibility to perform operations directly on compressed images.

2.1 Operations working on images in corner representation

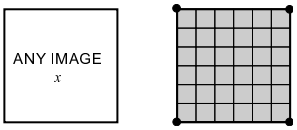
Shifting of the image (object) means adding value of the shift radiusvector to appropriate coordinate of each corner. The complexity depends linearly on number of corners.

Scaling corresponds to multiplication of each corner coordinate by a non-zero scale coefficient.

XOR is denoted as $\oplus$. $x \oplus y = z$, $z(i, j) = x(i, j) \oplus y(i, j)$. Let $\#_c(x)$ denote the number of corners in a picture x. The following holds $\#_c(z) \leq \#_c(z) + \#_c(y)$. The complexity of $\oplus$ depends on number of corners in both images.



NOT $\overline{x} = x \oplus 1$, where the number 1 represents the entirely black image. Four operations are needed only, i.e. the number of operations does not depend neither on the number of pixels nor number of corners.



OR, AND operations need more elaborate algoritms [1, 6] and we do not explain them here due to space constraints. The complexity depends on number of corners in both images.

Dilation, erosion can be effectively implemented in corner representation [3]. In our package, the dilation is implemented and erosion is performed via duality principle.

3 Connected components counting and labeling

Counting and labeling of connected components is in a separate module as algorithms for NOT, XOR and AND work on compressed image directly. Connected componenets are based on the region contours. Let us sketch the related algorithm.

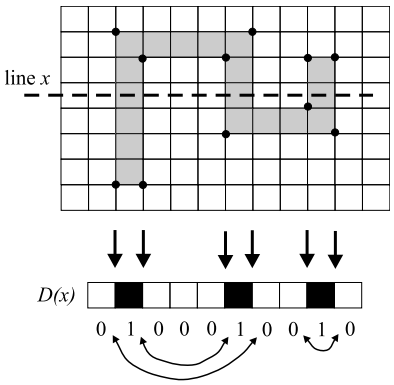


Figure 4: Data structure for counting process.

Lets us introduce an algorithm for connected components counting. Let m be number of lines in the image. A list $M(i)$, $i = 1, \ldots, m$, stores corners in the line i of the image. The 1D array $D(x)$ with the length equal to a number of columns is initially filled by zeros. The element of the array $D(x)$ is set to the color number corresponding to the color of a corner positioned above. The element should hold the i-th of the left most or right most (it depends if it is righ most or left most) corner of the same color. Let us assume that lines above the currently processed line i were already analyzed. The situation is illustrated in Fig. 4. The upper part shows the content of the image. From the point of view of the counting algorithm there are three regions for that it is not known how they are connected and they are assumed (provisionally) to be colored by labels 1, 2, 3. Of course, finally they will be all colored by one label. The corresponding corners are stored in the list M. The 1D array $D(x)$ shown in the bottom part in Fig. 4 tells how components of the image that was already analyzed are connected. The $D(x)$ stores only the left most and the right most column of each color. Assume a border point in a column x. Analysis of D provides information where the corresponding second end of the boundary is positioned. This process is denoted by arrows in Fig. 4.

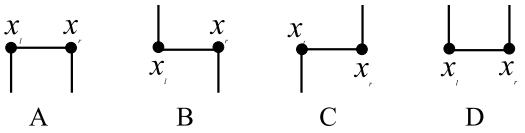


Figure 5: There are only four possible corner configurations while analyzing current line and array D.

We can explain the counting algorithm now. The image is processed line by line. A pair of corners is analyzed in each step of the algorithm. According to the content of the array D, i.e. the gathered information about connectivity, it is known how the currently processed line modifies D. There are only four possible cases that are illustrated in Fig. 5 and denoted as A, B, C, D.

while not last row **do**
Read a new pair of corners x_l, x_p.
Check the array D at the coordinates x_l, x_p.
case of situation

A: Increment the counter of continuous parts.

B: Fill D between x_l and x_p by the value $D(x_l)$.

C: Fill D between x_l and x_p by the value $D(x_p)$.

D: Clear D between x_l and x_p, i.e. fill it by zeros.

endcase
enddo

As can be seen from the sketch of the connected component counting algorithm, it needs only a single pass through the corner represented image M. Its time complexity is linear dependent on number of corners.

This algorithm does not treat the inner bouderies, i.e. regions with holes. For doing that, one more data structure is needed that stores whether the region has holes or not. Unfortunatelly, the linearity of the algorithm is lost in this point. The Schlesigner's implementation should be linear even for multiply connected regions.

The simple operations can be added to counting connected components, e.g. calculating region perimeter, area.

The algorithm for connected components labeling is an extension to the described counting. More details can be found in notes from [6].

4 Skeletonization in corner represented image

4.1 Modified definition of the skeleton

A skeleton of a binary object in a "continuous image" is traditionally defined as a union of centers of inscribed circles. Various modifications to the definition are proposed in mathematical morphology to cope with digital images and ensure that the homotopy of the original object and its skeleton is secured. The related algorithms either expel layers of the object (sequential thinning) or calculate maxima of a quench (distance) function [7].

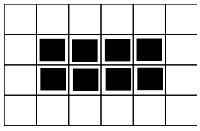


Figure 6: Ambiguity of the center of the object and of the thinning to width one.

Image in corner representation expressed by object outlines as a subset of points in a plane allows to avoid traditional problems in digital images, e.g. to be in the center or to be thinned to width one. The simple object in traditional pixel representation possesses such an ambiguity as is illustrated in Fig. 6. Schlesinger et. al suggested the concept of skeletonization in corner representation. It is explained in Russian elsewhere. Let us reference to the paper in English [4] and sketch basic ideas here.

Let the *image* $V \subset R^2$ be composed from black (object) point in an Euclidean plane. Every connected component is enclosed by a non-selfcrossing polygonal line that is composed from vertical and horizontal line segments only. A *maximal square* $S_m \subset V$ is a square inscribed to the object and there is no bigger inscribed square. The notion of the maximal square is illustrated in the left side in Fig. 7.

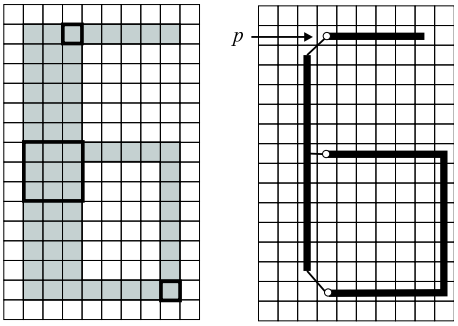


Figure 7: One of the maximal squares S_m is shown in the left part. The right part shows the basic skeleton (thick lines) that consists of centers of all possible maximal squares. The thin part constitutes the auxiliary skeleton.

The *basic skeleton* of the image V is defined as the set to central points of all maximal squares. Every connected component of the basic skeleton is either an isolated point or consists of one or more horizonatal/vertical line segments. The basic skeleton is shown as a thick lines in Fig. 7 at the right side. Two types of ends of basic skeleton segments are distinguished. The *closed end* corresponds to the point that closes the object in that end, i.e. the point does not belong to other bigger maximal square. The end is called the *open end* otherwise. Open ends are illustrated by empty circles in Fig. 7.

Let p be an open end of the segment of the basic skeleton. There is a maximal square S_m corresponding to the point p. It contains squares $S \subset V$ with the center in p. The maximal square attached to point p is shown as thick square in Fig. 7 at the right side.

The *auxiliary skeleton* is a set of additional line segments, each of them links the open end of the basic skeleton with the center of its attached maximal square. The auxiliary skeletons join basic skeletons belonging to one region (connected component). Auxiliary skeleton segments are shown as thin lines in Fig. 7 at the right side.

4.2 An idea of the skeletonization algorithm

The skeletonization algorithm explores the "divide and conquer" strategy. The region is decomposed by a *dissective square* into two regions. This process is repeated recursively until it is trivial to find a skeleton of the remaining regions.

Let q be a concave corner of the region R. Recall that notion of convex and concave corners was introduced in Section 2. A square $S_d \subset R$ is called a *dissective square*, see Fig. 8, if (a) one of the S_d vertices coincides with corner q, (b) one of the S_d diagonals coincides with the bisectrix (i.e. line dividing the angle of the corner to one half) of the corner q, and (c) there is no square S for which $S_d \subset S$ and conditions (a), (b) are satisfied simultaneously. One of two sides of the dissective square S_d not continuing q has a nonempty intersection with the region boundary at least. Let q^* be any point of this intersection. Let assume that q and q^* belong to the same connected component (region) G_c. Points q and q^* split the boundary G_c into two parts G_{c1}, G_{c2} and the boundary of the dissective square S_d into two parts G_{s1} and G_{s2}.

A region R is *decomposed* into two regions R_1 and R_2, see Fig. 8, if (a) $R_1 \cap R_2 = S_d$ and (b) the boundary of each new region contains only one of the parts G_{c1}, G_{c2} and only one part of the G_{s1}, G_{s2}. The *connector* links centers of two maximal squares containing S_d and belonging regions R_1, R_2, respectively.

Three following statements hold about region decomposition. The basic skeleton of the region R is a union of basic skeletons of regions R_1 and R_2. The auxiliary skeleton of the region R is a union of auxiliary skeletons of regions R_1, R_2 and a third set containing the connector only. Area (or perimeter) P of the region is equal to $P_1 + P_2 - P_d$, where P_1, P_2, P_d are areas (or perimeters) of regions R_1, R_2 and dissective square S_d, respectively.

A region is called a *simple region* if its boundary does not contain concave corners D1, H0, cf. Fig 2 and does not intersect with vertical sides of dissective squares corresponding to concave corners F1, G1. The simple region can be classified into three classes, see Fig. 9. The class 0 does not contain any

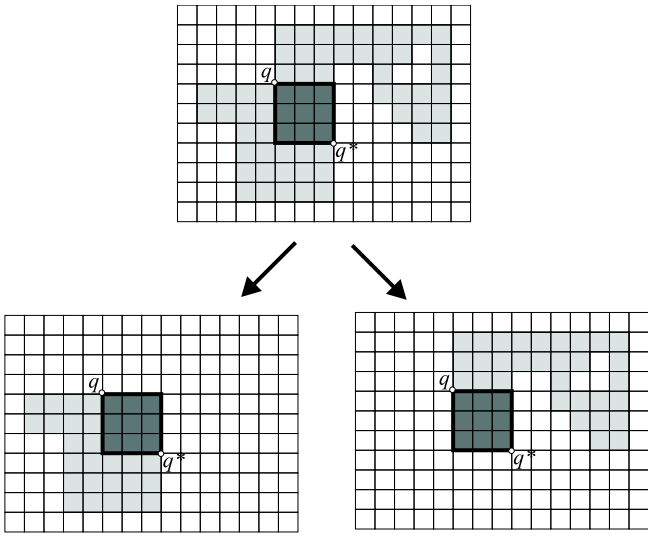


Figure 8: Decomposition of the region into two simpler components by the dissection square.

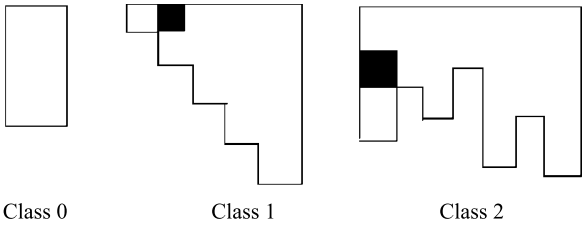


Figure 9: Classes of regions. Dissective squares are shown in black.

concave corners. Its skeleton is trivial, i.e. a line segment. The class 1 contains only concave corners F1. That one with the smallest x coordinate is selected and the dissective square corresponding to it is determined. The simple region is split into two ones. One region is a rectangle, i.e. class 0 and the other one remains in general the class 1 region. This procedure is repeated until all concave (F1) corners are removed. The class 2 does not contain concave corners D1, H0. There is a concave corner G1 in the class 2 region that is used to split the region.

The splitting procedure for the most general case containing also corners D1, H0 is a bit more complicated, see [4, 3].

Two approaches are possible. At first, the general region is split into set of regions of class 2. Then each region of class 2 is split into a set of regions of class 1, and finally the regions of class one are split into a seto of class zero regions. At second, which is more efficient, the general region is split again into class two regions. These are split into one or more simple regions and set of rectangles. Simple objects are split into rectangles.

The skeletonization algorithm in corner represented image does not depend on the thickness of the thinned regions. The computational complexity depends on the number of corners that is much smaller compared to number of region pixels in traditional skeletonization algorithms.

5 Modifications to the skeletonization algorithm

We did some changes to the skeletonization algorithms proposed in original work [4, 6, 3]. Instead of using class 1 regions, a general region is split into class 2 region. Afterwards, each class 2 region is split into rectangles and a simple object. The simple region is split into rectangles as well as we know from previous Section already. The result is set of rectangles. It is trivial to find their skeleton.

We implemented our own algorithm for dissecting objects of class two. It uses finite automaton with four states.

We created a graph storing topological relations which corresponds to dissective squares in the algorithm mentioned above. This graph tells how are skeletons of parts connected together by dissective squares. This graph is then used to create an output. An information stored in the graph allows to reconstruct the original binary image with all its regions, to draw the skeleton, dissective squares, and all rectangles which constitute the image. We did not use the exact definition fo the auxiliary skeleton as described in Section 4.1. We used just auxiliary lines oriented modulo 45 degrees to connect basic skeleton as this process is ambiguous.

If the skeleton is used for vectorization than its beautification is useful. The skeleton obtained from a scanned real raster image is typically broken to unnecessary many small line segments. In this context, beautification means approximating several smaller segments by longer line segments. Cartographers would call this process – generalization. The topological graph stores all needed information for beautification.

6 Package design and implementation

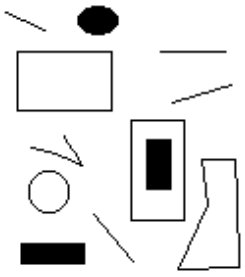


Figure 10: Input binary image 130 $\times$ 150 pixels with synthetic object.

The Corners package is implemented in Microsoft C$_{++}$. It is composed of six parts.

Coding/decoding to corner representation. This part accepts a raster image in a special inner format or reads .bmp files. Objects in the image are transformed to corner representation and can be stored in a file with extension .crn. The decoding from corner image to raster image works in opposite direction and provides a .bmp file.

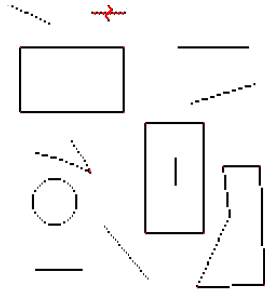


Figure 11: Found skeleton of synthetic objects.

Connected components counting and labeling allows the user to specify with which objects she/he intends to work.

Operations on a compressed picture, i.e. rotate, scale, shift, AND, OR, NOT, XOR, and morphological dilation, erosion.

Skeletonization provides the skeleton and represents it in a topological graph.

Skeleton beautification runs on the topological graph and finds a path segments in a skeleton image, i.e. segments where inner vertices are of degree 2. A skeleton smoothing can be performed during this phase.

Vectorization transfers data in the topological graph into the format readable by vector editors. Briefly said it is trying to interpolate the huge number of connected small lines by longer lines. Work on this module is in progress.

The executable program (still under development) running under Microsoft Windows is available for experiments at `http://cmp.felk.cvut.cz/~hlavac/Public/corners`. Use binary .bmp images as input.

7 Experiments

Let us show the experiment on synthetic data first. The input image is shown in Fig. 10. Its 1 bit raster version in .bmp format needs 3062 bytes. When transformed to corner representation only 300 corners are needed and occupy 463 bytes if stored on a disc in .crn (corners) binary format. The found skeleton is in Fig. 11.

The experiments with real data were performed as well. We used deliberately the same image of an engineering drawing (3D view of a ventilator assembly) as was used in original experiments by Kiev group [3] to allow comparison of implementations. The input binary raster image has size 1700 $\times$ 1248 pixels and occupies in .bmp format 6364938 bytes, see Fig. 12. The same image in corner representation needs only 69688 bytes which corresponds to the compression to 1.01%, see Fig. 13. The skeleton was calculated in the corner representation. The result is shown in Fig. 14. The 120 MHz Pentium with 64 Kbytes of operational memory was used for experiments. The conversion to corner representation took 3 seconds and skeletonization 4 seconds.

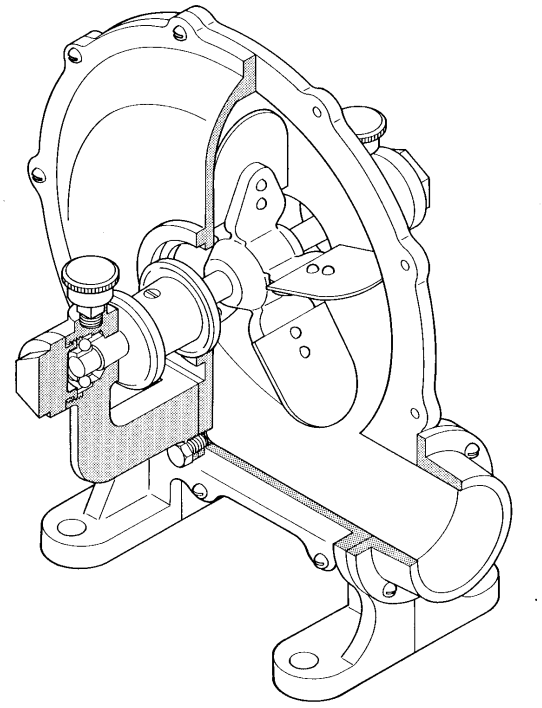


Figure 12: A view on a ventilator. The imput binary raster image.

8 Conclusions

The idea of corner representation was sketched in the paper first. The implementation of the toolbox allowing to process binary images in efficient corner representation was reported. The implementation carried out by group of studens in a software engineering project. The development of the toolbox is continued.

References

[1] C. L. Aleksandrova. Bystrodjstvujuij algoritm logieskoj filtracii binarnych izobraenij, in Russian, (Quick algorithm for logical filtering of binary images). In *Teoretieskije i prikladnyje voprosy raspoznavania izobraenij, (Theoretical and applied issues in Pattern Recognition*, pages 25–35. Akademia Nauk Ukrajinskoj SSR, Institut Kibernetiki, Kiev, Ukrajina, 1991.

[2] V. Hlaváč and J. Fojtík. Adaptive predictor for lossless image compression. *Computing*, 62(4):339 – 354, June 1999.

[3] V. Kiyko and M. Schlesinger. Fast dilation and skeletonization of compressed binary pictures. In H. H.J.A.M., editor, *Mathematical morphology and its applications to image and signal processing*, pages 347–354. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[4] V. M. Kiyko and M. I. Schlesinger. Width-independent fast skeletonization algorithm for binary pictures. *International Journal of Imaging Systems and Technology*, 3:222–226, 1991.

[5] M. I. Schlesinger. *Matematiceskie sredstva obrabotki izobrazenij, in Russian, (Mathematic tools for image processing)*. Naukova Dumka, Kiev, Ukraine, 1989.

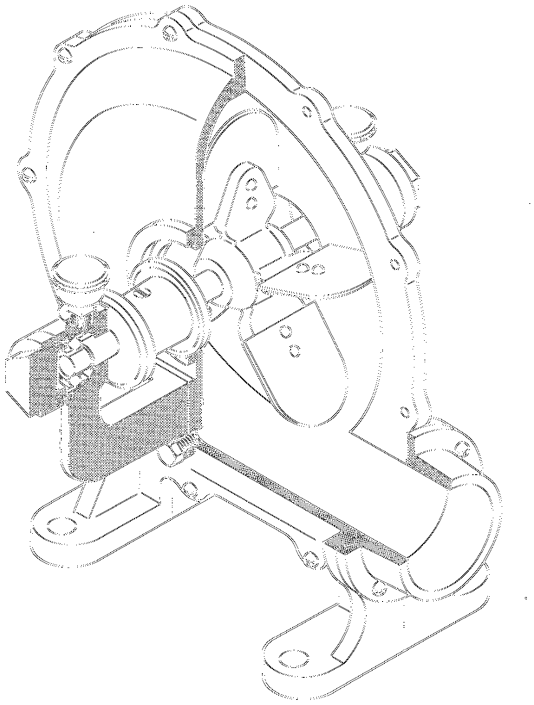


Figure 13: The ventilator represented by corners

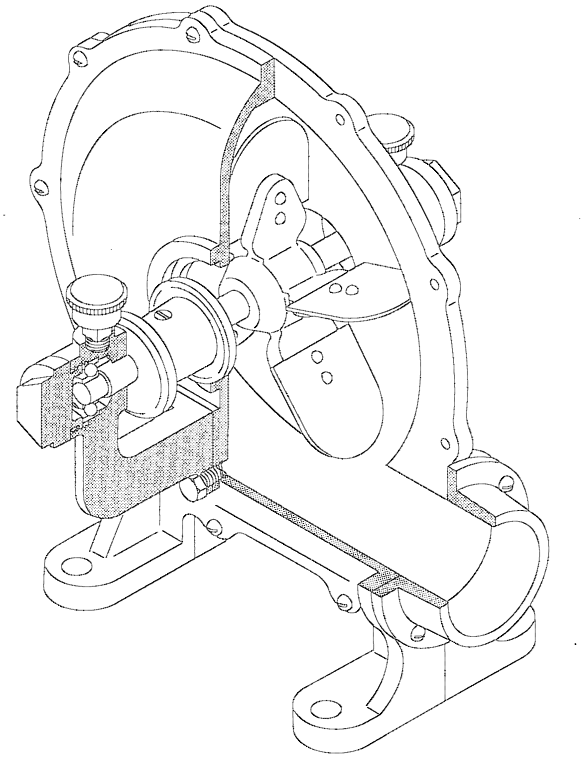


Figure 14: The skeleton of the ventilator image.

[6] M. I. Schlesinger. Pattern recognition. Series of 15 lectures held at the CTU Prague in spring 1996, 1996.

[7] M. Šonka, V. Hlaváč, and R. Boyle. *Image Processing, Analysis and Machine Vision*. PWS, Boston, USA, second edition, 1998.