Shape Extraction Framework for Similarity Search in Image Databases

Jan Klíma and Tomáš Skopal

Charles University in Prague, FMP, Department of Software Engineering
Malostranské nám. 25, 118 00 Prague, Czech Republic
irenicus@volny.cz, tomas.skopal@mff.cuni.cz

Abstract. The task of similarity search in image databases has been studied for decades, while there have been many feature extraction techniques proposed. Among the mass of low-level techniques dealing with color, texture, layout, etc., an extraction of shapes provides better semantic description of the content in raster image. However, even such specific task as shape extraction is very complex, so the mere knowledge of particular raster transformation and shape-extraction techniques does not give us an answer what methods should be preferred and how to combine them, in order to achieve the desired effect in similarity search. In this paper we propose a framework consisting of low-level interconnectable components, which allows the user to easily configure the flow of transformations leading to shape extraction. Based on experiments, we also propose typical scenarios of transformation flow, with respect to the best shape-based description of the image content.

1 Introduction

Similarity search in image databases [10] is becoming increasingly important, due to rapidly growing volumes of available image data. Simultaneously, the text-based image retrieval systems become useless, since the requirements on manual annotation exceed human possibilities and resources. The metadata-based search systems are of similar kind, we need an additional explicit information to effectively describe multimedia objects (e.g. structured semantic description, as class hierarchies or ontologies), which is not available in most cases.[1] The only practicable way how to search the vast volumes of raw image data is the content-based similarity search, i.e. we consider the real content of each particular raster image (e.g. a photography), where the images are ranked according to similarity to a query image (the example). Only such images are retrieved, which have been ranked as sufficiently similar to the query image. The similarity measure returns a real-valued similarity score for any two models of multimedia objects.

Unlike other similarity search applications, the task of highly semantic content-based search in image/video databases is extremely difficult. Because of generally unrestricted origination of a particular raster image, its visual content is

[1] The image search at images.google.com is a successful example of metadata-based engine, where the metadata are extracted from web pages referencing the images.

J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2007, pp. 89–102, ISBN 80-7378-002-X.

not structured and, on the other side, hides rich semantics (as perceived by human). The most general techniques providing extraction of distinguished features from an image are based on processing of low-level characteristics, like color histograms, texture correlograms, color moments, color layout (possibly considered under spatial segmentation). Unfortunately, the low-level features emphasize just local/global relationships between pixels (their colors, respectively), hence, they do not capture high-level (semantic) features. In turn, usage of the low-level features in similarity search tasks leads to poor retrieval results, which is often referred to as the "semantic gap" [10].

In real-world applications, a design of high-level feature extraction is restricted to the domain-specific image databases. For instance, images of human faces can be processed so that biometric features (like eyes, nose, chin, etc.) are identified and properly represented. Although domain-specific image retrieval systems reach high effectiveness in precision/recall, they cannot be used to manage heterogeneous collections, e.g. images on web.

1.1 Shape Extraction

The shapes (contours, bounded surfaces) in an image could be understood as a medium-level feature, since shape is an entity recognized also by human's perception (unlike low-level features). Moreover, shape is a practical feature for query-by-sketch support (i.e. a query-by-example where the "example" consists of user-drawn strokes), where extraction of colors or textures is meaningless.

Shape Reconstruction. The most common technique is to vectorize the contiguous lines or areas in the raster image. Prior to this, the image has to be preprocessed still on the raster basis (edge detection [17], smoothing, morphologic operations, skeletonization, etc.). The subsequent raster-to-vector transformation step follows (e.g. a binary mask is vectorized into a set of (poly)lines).

Naturally, we are not done at this moment, the hardest task is to filter and combine the "tangle" of short lines (as typically produced) into several (or even single) distinguished major shapes (polylines/polygons). This involves polyline simplification [11], removal of artifacts, line connection, etc. The most complex but invaluable part of shape reconstruction should derive the prototypical shape which is approximated by the vectorized information obtained so far.

Shape Representation & Similarity Measuring. Once we have sufficiently simplified shapes found in an image, we have to represent them in order to support measuring of similar shapes. The polygonal representation itself is not very suitable for similarity measuring, because of high sensitivity to translation, scale, rotation, orientation, noise, distortion, skew, vertex spacing/offset, etc. More likely, the raw shape is often transformed into a single vector or time series [5, 7, 9], where the shape characteristics are preserved but the transformation non-invariant characteristics are removed. The time series representations are usually measured by Euclidean distance, Dynamic time warping [4, 7], Longest common subsequence [15].

1.2 Motivation & Paper Contributions

As overviewed above, the process of shape extraction (starting from the raster image and producing a vector or time series) is very complex task, where there do not exist general recommendations about particular transformation/extraction steps. In this paper, we propose a component-based framework allowing to encapsulate and connect various image/vector-processing algorithms. Hence, a network consisting of many components can be easily created, through which a particular shape extraction scenario is configured. Following this framework, we have also implemented a catalogue of basic components which, when properly connected, can provide an effective environment for shape extraction experimentation. Finally, we present several domain scenarios for shape-extraction based on experimental results.

1.3 Related Work

Although we are aware of many existing image processing libraries, most of them lack support for end-user dataflow configuration or vector processing.

"Filters" [1] is a library of image, video and vector processing functions, based on idea of configurable filters that perform various tasks. Dataflow configuration is obtained by hardcoding or via python scripts.

"JaGrLib" [12] is a 2D/3D graphics library primarily aimed for educational purposes. JaGrLib offers both XML and GUI oriented dataflow configuration, but currently has limited shape extraction capabilities.

2 IVP Framework

The idea of Image and Vector Processing Framework [2] is to separate objects that usually figure in image processing (color bitmaps, grayscale bitmaps, binary bitmaps, gradient maps, vectors, polylines, etc.) and algorithms which work with these objects on an *input* → *output* basis. Each algorithm can be considered as a black box that expects certain input data and produces a defined kind of output data. With this point of view the whole shape extraction application reduces to a network of algorithms that send data to each other. An example of the idea is depicted in Figure 1.

This gives a view into the IVPF design: it's advantageous to code and store algorithms separately, the role of the client application is to allow user to specify which algorithms should be used, and also their *output* → *input* dependence. In the final effect, many specialized applications can be implemented (configured respectively) at high application level, just by specifying the algorithms, their settings and mutual dependencies.

2.1 Interface & Components

Each particular component class encapsulating[2] an algorithm (as mentioned above) must implement the `IIVPFComponent` interface in such a fashion that all

[2] The framework has been implemented in .NET framework 2.0.

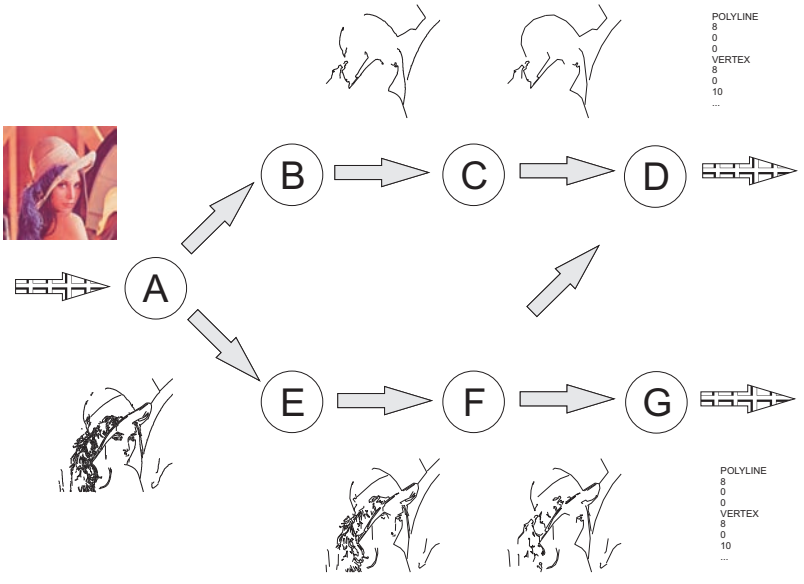

Fig. 1. Algorithms as black boxes: The example shows algorithms A-G which form an application with arrows representing their *output* → *input* dependence. A is an input algorithm that vectorizes the input image and sends the resulting vectors into the network. B-C and E-F-G are branches that transform their vector input somehow. D takes two vectorial inputs and chooses one, based on certain criteria. Then both D and G save their outputs in specified formats (to a file/stream/anything else).

public component features are accessible in a simple and transparent way. In particular, using `IIVPFComponent` interface the components are interconnected (port registration), and checked for compatibility.

Higher Level Functionality. At a higher application level, the components are just configured and their ports connected together to create a network. This can be done either via GUI or by loading configuration/connections from an XML file. During the whole network execution, all components are run starting from pure output components (no input ports) and propagating the intermediate data through the entire network to pure input components (no output ports). The whole approach enables to change the behavior of the network in two ways:

- by changing component(s) configuration
- by changing the structure of the entire component network

3 Component Catalogue

In this section we propose several components already implemented in the IVP framework. In order to ease the understanding, we use the following formal declaration of a component:

`type of input` → `component name (parameters)` → `type of output`

where the type of input/output we distinguish either `bitmap` (any 2D bitmap of color, intensity, gradient, etc.) or `vectors` (collection of polygons/polylines). The single arrow '→' means there is just a single type of connection to input/output supported, while the double arrow '⇒' supports several types of input/output[3]. The parameters declared in parentheses are component-specific, thus they have to be configured by the user.

3.1 Image Processing Components

The first class provides components serving as the basic-processing tools, which do eliminate resolution dependencies, filter the noise, and separate pixels within a specified range of colors.

Image Loader Component
`ImageFromFile (FileName)` ⇒ `bitmap` (of colors) + `bitmap` (of intensities)

To process an image, it must be loaded from a file first. During this phase grayscale image representation is computed and offered simultaneously.

Image Resample Component
`bitmap` (colors) → `ImageResample (ResamplingType, DesiredSize)` ⇒ `bitmap` (colors) + `bitmap` (intensities)

The input image might be either too small or very large for the sake of further processing. With this component it's easy to resize it suitably using Nearest Point, Bilinear and Biquadratic resampling.

Thresholding Component
`bitmap` (colors) → `ImageThresholding (RGBUpper, RGBLower)` → `bitmap` (binary)

There exist special types of images like maps or drawings where it makes little sense to do full edge detection. Instead, certain parts of interest can be extracted by simple interval thresholding.

Gaussian Smoothing Component
`bitmap`(intensity)→`GaussianIntensityFilter(WindowSize,Sigma)`→`bitmap`(intensity)

In noisy images where one would expect problematic edge detection, smoothing step is required. Gaussian smoothing is a well-established method and the implementation allows to configure both the Sigma parameter of the Gaussian function and the window size.

[3] Naturally, an output port of one component can be connected to input ports of multiple components (providing we obey port compatibility).

3.2 Edge Detection Components

For edge detection, the Canny operator [8] was chosen as a main approach, as it is acceptably stable and configurable. The edge detection is performed in multiple stages, starting on an intensity (grayscale) bitmap, which usually involve

1. Source image smoothing (typically by Gaussian convolution)
2. Gradient approximation using first derivative operator
3. Non-maximum suppression
4. Hysteresis thresholding to identify edge pixels

The Canny operator is formed by a chain of components (the latter described below): `GaussianIntensityFilter` → `GradientOperator` → `NonMaximaSuppression` → `HysteresisThresholding`. For an example of the dataflow see Figure 2.

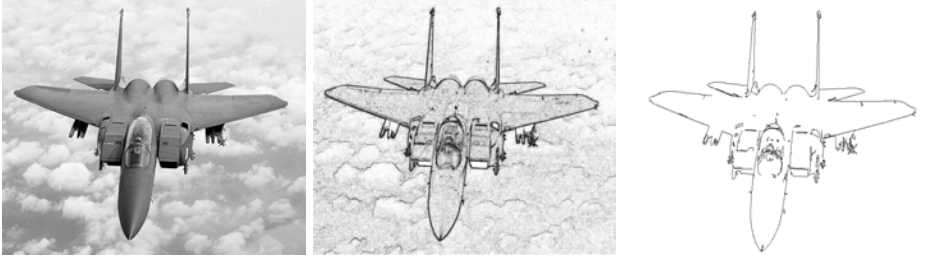

Fig. 2. An example showing input image, image's gradient map and marked edge pixels.

Gradient Operator Component
`bitmap` (intensity) → `GradientOperator (OperatorType)` → `bitmap` (gradients)

Uses simple first derivative operators (Sobel, Prewitt, Roberts-Cross) to obtain local gradient magnitudes and directions (in 45 degree steps) for each pixel.

Non Maximum Suppression Component
`bitmap` (gradients) → `NonMaximaSuppression` → `bitmap` (gradient)

Gradient map obtained by first derivative operator often contains thick regions with high gradient magnitude but to extract edges one would like areas with high gradient magnitude to be as much thin as possible. Non-maximum suppression archives this by ignoring pixels where the gradient magnitude is not maximal in the gradient direction.

Hysteresis Thresholding Component
`bitmap` (gradients) → `HysteresisThresholding (Lower, Upper)` → `bitmap` (binary)

Based on two thresholds gradient map is traced and edge pixels are extracted. Each pixel with gradient magnitude greater than the Upper threshold is marked as edge immediately. Remaining pixels are marked as edges only if they have their gradient magnitude greater than the Lower threshold and if they are connected to some existing edge pixel chain.

3.3 Binary Image Processing Components

The following components process binary bitmap inputs (e.g. obtained from the edge detection). These components could be used to simplification of contours.

Erosion And Dilatation Component
`bitmap`(binary)→`ErosionDilatation (OperationType, MaskType)`→`bitmap`(binary)

Refinement of binary images is often needed (to close gaps, round curved objects, etc.). The operators of erosion, dilatation, opening and closing (combined with a properly chosen mask) are usually a good choice to handle some input image defects.

Thinning Component
`bitmap` (binary) → `Thinning (ThinningType)` → `bitmap` (binary)

Thinning components that implement Stendiford [14] and Zhang-Suen [16] thinning algorithms are handy when it comes to polish results given by edge detection, or when there is a need to turn thick objects into one pixel thin lines. A staircase removal to refine lines contained within the binary image also fits in this category of algorithms. An example of the thinning process is given in Figure 3 (the first two images).

Fig. 3. Input binary image along with its thinned form and refined vector result containing $n = 18$ polylines.

Contouring Component
`bitmap` (binary) → `Contouring` → `bitmap` (binary)

In some cases (when the binary image contains thick objects), an information about contour is more valuable than the object's thinned form. Implemented method is based on approach found in [3] although it uses complete image matrix instead of run length encoding as the binary image representation.

Vectorization Component
`bitmap` (binary) → `Vectorization` → `vectors`

The vectorization component is responsible to turn binary image with marked edge pixels into a set of vectors (polylines). It is based on approach mentioned by [3] or [6].

First, the input binary mask is went through by shifting a 3x3 window over every pixel in order to mark critical points. Those are either endpoints (pixels with only one neighbor within the 3x3 window) or intersections (pixels with more than 2 neighbors). In second phase all polylines between critical points are traced. Another pass through

the image is needed then to identify closed polylines that were left untouched by the previous step. The resulting pixel chains are turned immediately into polylines along with junction and connectivity information (i.e. with the topology).

3.4 Vector Processing Components

Once we get a vectorized form of shape, we move to waters of geometry and graphs algorithms. Although we got rid of the raster information, we now face a tangle of (poly)lines to be meaningfully managed.

Polyline Simplification Component
`vectors → PolylineSimplification(Type, Error) → vectors`

The polylines obtained from vectorization usually carry much more information than required. It also happens that there are undesired irregularities in polylines caused by straightforward vectorization. Hence, a polyline simplification is needed. Multiple approaches are implemented, including those of Douglas-Peucker [11] and Rosin-West [13].

Artifact Removal Component
`vectors→ShortVectorRemoval(ArtifactsType,ArtifactCharacteristics)→vectors`

Aside from polyline simplification, the resulting vector image contains artifacts at higher logical level. For a list list o typical artifacts see Figure 4. Artifact removal component handles these artifacts based on configuration and produces result with reduced noise and longer (more meaningful) vectors, as shown in Figure 4.

Fig. 4. The first picture gives example of the most typical small artifacts – **a.** unconnected polylines **b.** insignificant cycles **c.** 1-connected polylines **d.** spurious loops **e.** insignificant junction connections. The second picture shows an input vectorized image. On the third picture there is output of the artifact removal component (configured to ignore short artifacts) and Douglas-Peucker polyline simplification algorithm.

Iterative Pruning
`vectors → IterativePruning (UpperBound) → vectors`

One of the goals of vector processing is to find a given number of the most significant vectors to represent the original image. With this component an experiment was made to find if a metric based on vector length is a good criterion for selection the most significant vectors.

The algorithm (our original design) works as follows: An upper bound is selected by user that represents the maximum number of vectors to obtain. First, all vectors

Fig. 5. The first picture shows line drawing containing $n = 12,617$ polylines made of $m = 38,433$ line segments. In second picture is the result of vector removal component (with *upperbound* = 20) combined with Douglas-Peucker simplification. The output contains $n = 19$ polylines with $m = 473$ line segments.

are sorted with respect to their lengths. The algorithm works in iterations and tends toward the desired number of vectors. In order to guarantee convergence, a half of the vectors are expected to be thrown away in each iteration (the number of vectors to throw away can be easily configured as well, giving slower of faster convergence).

Which vectors should be thrown away is decided upon their lengths(short vectors are considered first) and upon the knowledge similar to that used in Artifact Removal Component (most probable artifacts are thrown away first). A typical output is depicted in Figure 5.

Polyline Connection Component

`vectors→PolylineConnection(Scenarios,ScenariosCharacteristics)→vectors`

It happens (especially in edge detection) that a significant polyline gets divided into multiple parts as a result of inaccurate thresholding, noise or overlap.

Three phase algorithm is employed to join parts together into bigger entities. The first phase operates on polylines that have their endpoints very close to each other and can be joined together without additional heuristics. Second phase takes care of larger gaps between endpoints with respect to multiple criteria (vector length, distance, orientation, etc.). Third phase tries to handle disconnected corners. Figure 6 features typical scenarios of these three stages. All phases are configurable and optionally offer many important concepts like double sided check, identical angle orientation check etc. Many of these concepts are mentioned in [6].

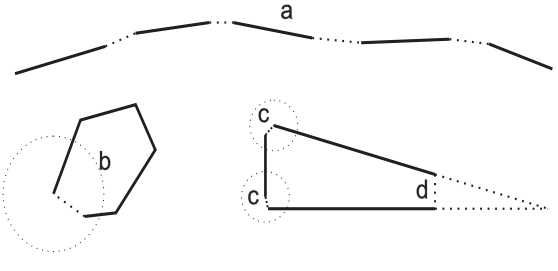

Fig. 6. An example demonstrating typical scenarios when connecting polylines: **a.** connection of lines based on their orientation, angle, ... to create one longer polyline **b.** closing disrupted cycles **c.** connecting near endpoints to form a corner **d.** connecting far endpoints to form a corner, guessing corner shape.

3.5 Vector Output Components

The last group of components provides an output to external storage (now file). Besides an one-to-one export to a well-known format (like WMF, DXF), components in this class cover also secondary feature extraction techniques needed for particular task – typically representations for subsequent similarity measuring/search.

Vectors Output Component
`vectors → VectorToFile(FileName, Format)`

This component saves its vector input into a specified format (DXF, textfile, etc.).

Angles Extraction Component
`vectors → AnglesToFile(FileName, AngleType, NumberOfAngles)`

A specialized feature component transforming polylines to (time) series. Each polyline is normalized first by splitting it to a number of parts of the same length. A set of angles is then saved to the output. The angles can be measured as the smaller angles between each pair of successive line segments, as clockwise (or counter-clockwise) angles between them, or as angles between each line segment and the X-axis.

4 Domain Scenarios

The components of the previously introduced catalogue have been designed in order to easily create scenarios suitable for extraction of simple shapes. Hence, we are primarily interested in simplified representation of shapes inside an image, rather than in a comprehensive image-to-shape transformation preserving as much information as possible.

A simplified shape could serve as a descriptor involved in measuring similarity of two images (based on shapes found inside). The requirement on small-sized descriptor is justified by handling the shape information by similarity measure. Since the similarity measure is supposed to be evaluated many times on entities of a huge image database, the similarity measuring should be as fast (yet precise) as possible. However, this goal can be achieved by a smart shape extraction providing prototypical descriptors. An image represented by a single (or very few) polylines/polygons limited to several tens or hundreds of vertices (say up to 1 kilobyte) – this is our desirable descriptor.

On the other side, we are aware of the difficulties when trying to establish such a simple descriptor. Therefore, we have performed experimentation on various images (photography, drawing, etc.) and tried to assemble several configurations of component networks (called domain scenarios) which gave us the best results for a particular image type (with respect to desirable descriptor properties). In the following sections we present three such scenarios.

4.1 Drawing

As for the drawings, the vectorization task is slightly simpler thanks to the fact that the source image contains the desired information in an easily identifiable form (monochromatic strokes describing shapes, colored areas in case of cartoons, etc.). The extracted layer or layers described by binary images can be further processed by means of thinning (in case of strokes) or contour extraction (in case of thick areas). The result often

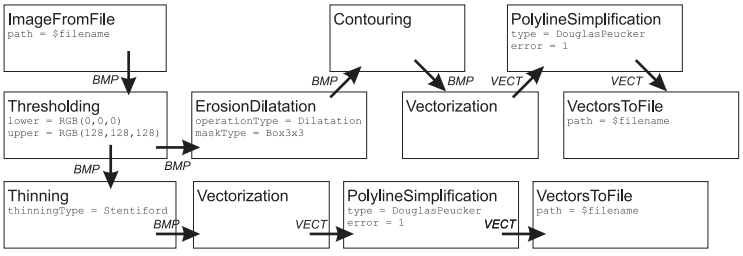

Fig. 7. Scenario 1 – Drawing.

embodies only a low level of noise and can be directly used as is or further simplified. In Figure 7 see the scheme of component configuration and interconnection under `Drawing` scenario. Note that the two branches lead to two different types of shape extraction (skeleton and contour). For an example of data flow regarding to the `Drawing` scenario, see Figure 8.

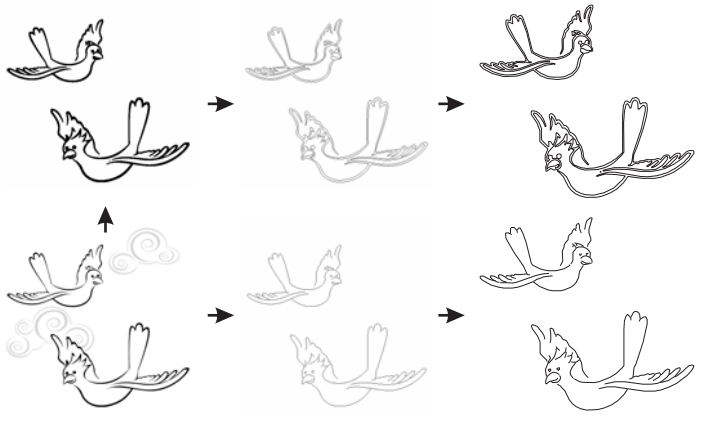

Fig. 8. Scenario 1 – Drawing – Example of data flow.

4.2 Simple Object

For high contrast images containing unsophisticated shapes, the edge detection alone is a reliable way to extract required feature information. When this is known, the artifact removal is a relatively safe operation without the risk of removing important features. A reconnection of disconnected lines (which follows then) almost completely reconstructs the full shape information. Finally, the polyline simplification should be done to straighten jagged lines and minimize the produced number of line segments. In Figure 10 see the scheme of component configuration and interconnection under the `Simple object` scenario. For an example of data flow, see Figure 10.

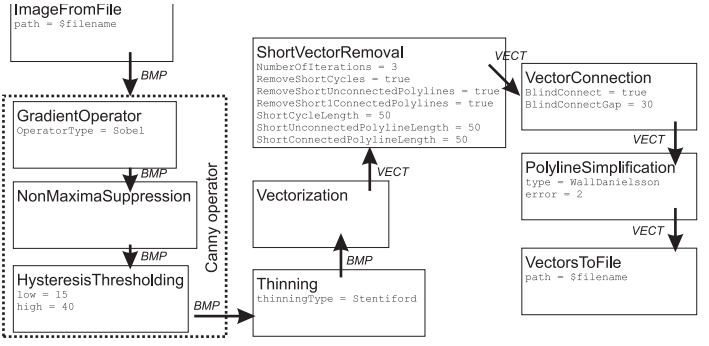

Fig. 9. Scenario 2 – Simple object.

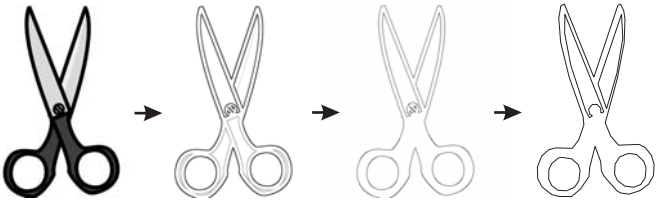

Fig. 10. Scenario 2 – Simple object – Example of data flow.

4.3 Complex Scene

In real-world images (photos), the edge detection cannot guarantee getting clean shapes, on the contrary there are usually huge amounts of false detected or unwanted edges. The iterative pruning is supposed to take care of most of the "trash" in the vector output and even then, maximum effort must be directed into connecting disrupted polylines, corner detection and polygonal approximation. In Figure 11 see the scheme of component configuration and interconnection under the `Complex scene` scenario. For an example of data flow, see Figure 12.

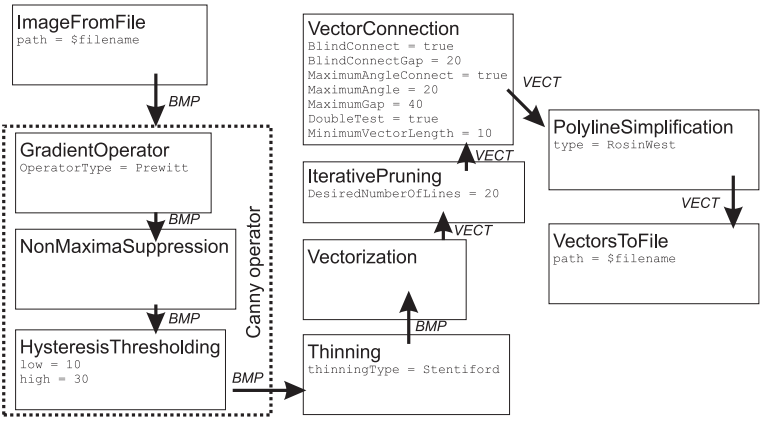

Fig. 11. Scenario 3 – Complex scene.

Fig. 12. Scenario 3 – Complex scene – Example of data flow.

5 Conclusions & Future Work

In this paper we have presented a highly configurable framework for shape extraction from raster images. Based on the framework, we have proposed a catalogue of components, which have been designed to be easily configured into a network. Based on experiments, we have recommended three domain scenarios for extraction of simple shapes, in order to create useful descriptors for similarity search applications.

In the future we would like to investigate similarity measures suitable for shape-based similarity search. An extraction of simple prototypical shapes from images (as proposed in this paper) is crucial for similarity measuring, so it is an unavoidable step when trying to "bridge the semantic gap" in image retrieval. Furthermore, we would like to automate the scenario recommendation process (where each component in scenario evaluates the goodness of what it produces), resulting in a kind of unsupervised extraction technique.

Acknowledgments.

This research has been partially supported by GAČR grant 201/05/P036 provided by the Czech Science Foundation.

References

1. Filters library (available at `http://sourceforge.net/projects/filters/`).
2. Image and vector processing framework (available at `siret.ms.mff.cuni.cz`).
3. S. V. Ablameyko. *Introduction to Interpretation of Graphic Images.* SPIE, 1997.

4. T. Adamek and N. O'Connor. Efficient contour-based shape representation and matching. In *MIR '03: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval*, pages 138–143, New York, NY, USA, 2003. ACM Press.
5. T. Adamek and N. E. O'Connor. A multiscale representation method for nonrigid shapes with a single closed contour. *IEEE Trans. Circuits Syst. Video Techn.*, 14(5):742–753, 2004.
6. P. Altman. Digitalization of map. Master's thesis, Charles University, Department of Software and Computer Science Education, 2004.
7. I. Bartolini and M. Patella. Warp: Accurate retrieval of shapes using phase of fourier descriptors and time warping distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(1):142–147, 2005. Member-Paolo Ciaccia.
8. J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.
9. A. Cardone, S. K. Gupta, and M. Karnik. A survey of shape similarity assessment algorithms for product design and manufacturing applications. *Journal of Computing and Information Science in Engineering*, 3(2):109–118, 2003.
10. V. Castelli and L. D. Bergman, editors. *Image Databases : Search and Retrieval of Digital Imagery*. Wiley-Inter., 2002.
11. D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a line or its caricature. *Canadian Cartographer*, 10(2):112–122, 1973.
12. J. Pelikán and J. Kostlivý. Jagrlib: Library for computer graphics education. In *WSCG (Posters)*, pages 125–128, 2004.
13. P. L. Rosin and G. A. W. West. Segmentation of edges into lines and arcs. *Image Vision Comput.*, 7(2):109–114, 1989.
14. F. Stentiford and R. Mortimer. Some new heuristics for thinning binary hand-printed characters for ocr. *IEEE Transactions on Systems Man and Cybernetics*, 13(1):81–84, 1983.
15. M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. Keogh. Indexing multidimensional time-series with support for multiple distance measures. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 216–225, New York, NY, USA, 2003. ACM Press.
16. T. Y. Zhang and C. Y. Suen. A fast parallel algorithm for thinning digital patterns. *Commun. ACM*, 27(3):236–239, 1984.
17. D. Ziou and S. Tabbone. Edge detection techniques - an overview. *International Journal of Pattern Recognition and Image Analysis*, 8:537–559, 1998.