Multimed Tools Appl (2014) 73:2033–2062
DOI 10.1007/s11042-013-1689-0

Sketch-based image retrieval using keyshapes

Jose M. Saavedra · Benjamin Bustos

Published online: 7 September 2013

Abstract Although sketch based image retrieval (SBIR) is still a young research area, there are many applications capable of exploiting this retrieval paradigm, such as web searching and pattern detection. Moreover, nowadays drawing a simple sketch query turns very simple since touch screen based technology is being expanded. In this work, we propose a novel local approach for SBIR based on detecting simple shapes which are named *keyshapes*. Our method works as a local strategy, but instead of detecting *keypoints*, it detects *keyshapes* over which local descriptors are computed. Our proposal based on *keyshapes* allow us to represent the structure of the objects in an image which could be used to increase the effectiveness in the retrieval task. Indeed, our results show an improvement in the retrieval effectiveness with respect to the state of the art. Furthermore, we demonstrate that combining our *keyshape* approach with a Bag of Feature approach allows us to achieve significant improvement with respect to the effectiveness of the retrieval task.

Keywords Sketch-based image retrieval · Content-based image retrieval · Local descriptors · Local matching

J. M. Saavedra (✉) · B. Bustos
PRISMA Research Group, Department of Computer Science, University of Chile,
Av. Blanco Encalada 2120, Santiago, Chile
e-mail: jsaavedr@dcc.uchile.cl

B. Bustos
e-mail: bebustos@dcc.uchile.cl

J. M. Saavedra
ORAND S.A., Santiago, Chile
e-mail: jose.saavedra@orand.cl

1 Introduction

In the last years, image retrieval has become a very relevant discipline in computer science owing mainly to the advances in imaging technology that has facilitated capturing and storing images. In a content-based image retrieval system (CBIR), an image is required as input. This image should express what the user is looking for, but the user frequently does not have an appropriate image for that purpose. Furthermore, the absence of such a query image is commonly the reason for the search [9]. An easy way to express the user query is using a line-based hand-drawing, a *sketch*, leading to the *sketch-based image retrieval* (SBIR). In fact, a sketch is the natural way to make a query in applications like CAD or 3D model retrieval [12].

A few authors have addressed image retrieval based on sketches. Some of these works are *Edge Histogram Descriptor* (EHD) [26], *Image Retrieval by Elastic Matching* [8], *Angular partitioning of Abstract Images* [5], *Structure Tensor*[9], and *Histogram of Edge Local Orientations* [23]. Recently, Eitz et al. [11] have presented results applying the Bag of Features (BoF) approach for the SBIR problem. Some of these methods will be described in the next section.

The main contribution of this work is to propose a novel local method based on detecting *keyshapes*. Our method takes into account structural information by means of the *keyshapes*, and local information by means of local descriptors that are also proposed in this work. Furthermore, based on the stroke's information, we present a novel strategy for detecting *keyshapes* that avoid efficiency issues that methods like those based on Hough Transform [29] undergo.

Our experimental results show a slight improvement in the retrieval effectiveness with respect to the state of the art. Nevertheless, owing to our method is based on a different feature (the structure of the objects) from those used by current methods, a combination of our method with a current leading method yields a significant improvement in the retrieval effectiveness.

In this paper, we show that a combination of our approach with the Bag of Feature (BoF) approach proposed by Eitz et al. [11] allows us to achieve significant improvement which is validated by a statistical test. Specifically, the combined method increases the retrieval effectiveness in almost 22 % of the effectiveness reported by the BoF approach.

The rest of this paper is organized as following. Section 2 describes the current methods for SBIR. Section 3 describes in detail the proposed method. Section 4 presents the experimental evaluation. Finally, Section 5 presents conclusions.

2 Related work

SBIR approaches can be classified as global or local techniques. Some interesting approaches falling in the global category are Edge Histogram Descriptor EHD [26], Angular Partitioning of Abstract Images (APAI) [5], and Histogram of Edge Local Orientation (HELO) [23].

Edge Histogram Descriptor (EHD) was proposed in the visual part of the MPEG-7 [21] and was improved by Sun Won et al. [26]. The goal is to get a local distribution of five types of edges (vertical, horizontal, diagonal 45°, diagonal 135°, and no direction)

from local regions of the image. The concatenation of local distributions composes the final descriptor.

Another important work on SBIR was presented by Chalechale et al. [5]. This approach is based on angular partitioning of abstract images (APAI). The angular spatial distribution of pixels in the abstract image is the key concept for the feature extraction stage. The method divides the abstract image into eight angular partitions or slices. Then, it uses the number of edge points falling in each slice to make up a feature vector.

Histogram of Edge Local Orientations HELO, proposed by Saavedra and Bustos [23], showed an improvement on the retrieving performance in SBIR. HELO computes a K-bin histogram based on local edge orientations. To get the HELO feature vector, first the sketch is divided in a $W \times W$ grid. Second, an edge orientation is estimated for each cell in the grid. Third, a 72-bin histogram is computed using each computed edge orientation. Finally, Manhattan distance is used to measure dissimilarity between histograms.

In the case of local techniques, these commonly represent a sketch by a set of feature vectors. Although these techniques are slower in time than the global ones, these may use local properties and structural information that could lead to better retrieving performance. Relevant approaches falling into this category are Shape Context [1] and STELA [24]. In addition, local techniques are characterized by supporting partial match and by working very well when objects are partially occluded.

Shape Context, proposed by Belongie et al. [1], is a local approach for measuring similarity between shapes. In *Shape Context*, sketches are represented by a set of points sampled randomly from the stroke points. The sampled points are then used as reference to compute a set of shape feature vectors. The shape context feature vector describes the distribution of the rest of the sampled points with respect to a given point by a log-polar histogram.

Structure Local Approach (STELA), proposed by Saavedra et al. [24], was originally applied for Sketch-based 3D object retrieval. STELA transforms a sketch into a set of *keyshapes* forming the whole sketch. STELA uses straight lines as *keyshapes*. Then, each *keyshape* is regarded as reference to compute a local feature vector. The final process is to match different feature vectors from two sketches. To this end, STELA applies the Hungarian Method using χ^2 test statistics as the cost function.

Another kind of technique is the Bag of Features (BoF) approach that uses local descriptors to obtain a codebook by means of a learning process. Recently, Eitz et al. [11] showed that this technique outperforms the known methods. The problem with BoF is that it does not support partial matching due to the localization information loss.

In the context of the BoF approach, Hu et al. [15, 16] proposed to use a Gradient Field image (GF) over which local descriptors are computed. The local descriptors are obtained by a variation of the HOG approach [7]. The local descriptors computed over the complete image database are clustered to obtain a codebook of approximately 1000 codewords. Although, the authors show good results over a small database, computing the GF is a time consuming process which requires to solve a sparse system of linear equations where the number of unknown terms is the order of the size of the input image.

In addition, a proposal for dealing with a large database was proposed by Cao et al. [4]. This method is based on the Chamfer Distance [2, 25] to measure dissimilarity between a sketch and an image. This approach does not show how to deal with position or scale variations although the authors present an interesting indexing method based on the inverted index structure.

Another approach for the SBIR problem is based on converting the input sketch into a regular image with color an texture [6, 10]. This conversion process is known as image montage. After applying the montage process, the SBIR problem is reduced to the classical CBIR problem in which an example image required as input is the result of the montage process. The montage based image retrieval leads to an expensive process, owing mainly to the additional process to convert the image into a regular image.

In this work we propose a novel local method for retrieving images given a simple sketch as input. Different from STELA, our proposal detects many types of keyshapes and is applied in the context of image retrieval. These keyshapes could be lines, arcs, elliptical shapes, just to mention a few. We show that our approach, based on *keyshapes*, in combination with the BoF approach outperforms significantly the state-of-the-art approaches.

3 Keyshape based approach

Sketches are characterized by representing the structural components of an object instead of representing color or texture information. For instance, when a person is asked to make a simple drawing of a teapot, he or she will probably draw three components: the body, the spout, and the handle like one of the pictures depicted in Fig. 1. In addition, the absence of color and texture information in sketches may cause the retrieval process to become a difficult task. This fact also means that techniques thought to work on regular images do not work appropriately with sketches. Therefore, in this section we present a novel method for retrieving images using a sketch as query. Our proposal is characterized principally by exploiting the structural information provided by sketches.

Definition 1 The object structure is the distribution of the parts that compose such an object.

From the Definition 1, we extract two relevant terms: (1) the component of an object, and (2) the distribution of the components.

1. *Component*: The components of an object are difficult to define because these exist in diverse scales. However, we will define a component from the geometric

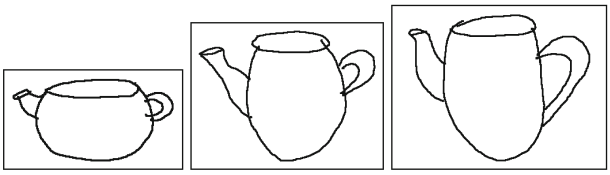


Fig. 1 Examples of hand drawings of a teapot

perspective. Therefore, a component is a simple geometric shape like an arc, an ellipse, a circle, a triangle, a rectangle, a square, or simply a straight line. In this way, the teapot showed in the Fig. 1 may be decomposed in an arc representing the handle, two arcs representing the spout, and an ellipse representing the body.

2. *Distribution*: The distribution of the components describes the spatial relationship between such components. In this way, using the teapot example again, the distribution should point out that the handle and the spout are located on the sides of the body; one on the left and the other on the right side.

To our knowledge, the methods proposed for the sketch based image retrieval do not exploit the structural property of sketches. The current methods are based on edge point distribution or edge point orientations which do not appropriately represent the structural components of the objects on the image. Furthermore, the interest point approach [27] in the computer vision field does not represent components with the semantic level as we are defining here. Moreover, the local region around *keypoints* could not be discriminating enough since sketches are simple line-based drawings.

The main contribution of this work is to propose a novel method for sketch-based image retrieval that takes advantage of the structural property of objects appearing in an image. Representing sketches by means of their structural components brings up the following advantages:

- Structural representation allows methods to represent objects on a higher semantic level which is reflected in the increment of the retrieval effectiveness.
- Structural representation allows methods to handle a smaller number of components with respect to the case of using an interest point approach. This leads to a more efficient matching step.

Our proposal deals with the structural property of objects by means of detecting simple geometric shapes that we call *keyshapes* which represent the structural components of objects. In addition, we propose two local descriptors computed over each detected keyshape. These descriptors represent the spatial distribution of the structural components. In short, our proposal exploits the structural property of an object describing the distribution of the parts composing it.

In addition, owing to our method is based on a different feature (the structure) from those used by current methods, a combination of our method with a method that has showed good results would lead to a significant improvement in the retrieval effectiveness. In this paper, we experimentally demonstrate such a property of our proposal.

Definition 2 A *keyshape* is a simple geometric shape that in conjunction with other simple geometric shapes composes a more complex object. Examples of a keyshape may be a circle, an ellipse, a square, a line, among other.

As showed in Fig. 2, our proposed technique consists of three stages (1) **keyshapes detection**, that allows us to detect simple shapes from an input image, (2) **local descriptor computation**, that allows us to locally represent the spatial relationship between a reference shape with respect to the others, (3) **matching**, that computes a cost value after setting some relation between two sets of local descriptors. In the following sections we will describe each one of these stages in detail.

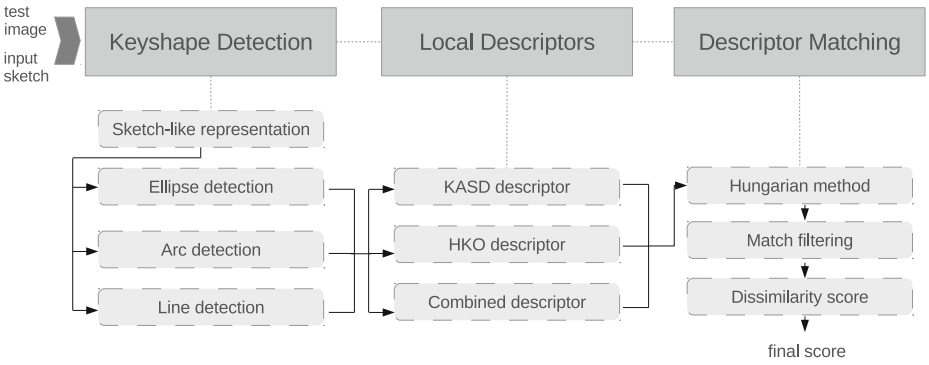


Fig. 2 A graphical representation of the stages involved in our SBIR approach

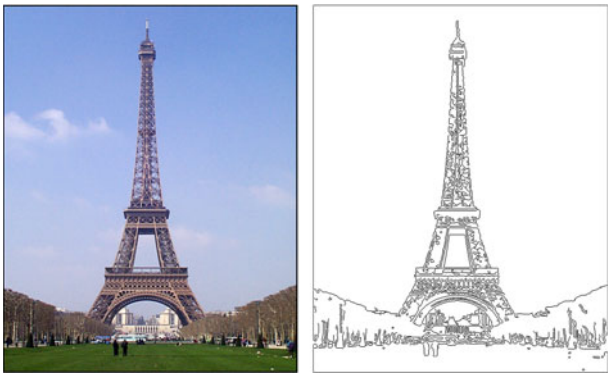
3.1 Keyshapes detection

In this section, we describe the first stage of our proposal. This stage involves, mainly, obtaining a sketch like representation, in particular from test images that are not sketches by themselves, and detecting a set of *keyshapes* that will be used for computing local descriptors.

3.1.1 Sketch-like representation

We will detect keyshapes from the input sketch (the query) and the images from the database (test images). In order to compare a sketch with a test image we require to transform a test image into a sketch-like representation. In particular, we need to transform the test images into sketches since they are regular color images. A simple way to carry out this transformation is by using an edge detection procedure. To this end, we use the Canny operator [3]. We prefer the Canny method than other methods like the *Berkeley boundary detector* [20] because of two reasons: (1) Canny allows us to get edge pixels accurately, and (2) computing Canny is much faster than the Berkeley's approach. In Fig. 3 we depict an image and its edge map representation computed using the well known Canny approach with $\sigma = 0.3$.

Fig. 3 Simple Canny edges of a test image



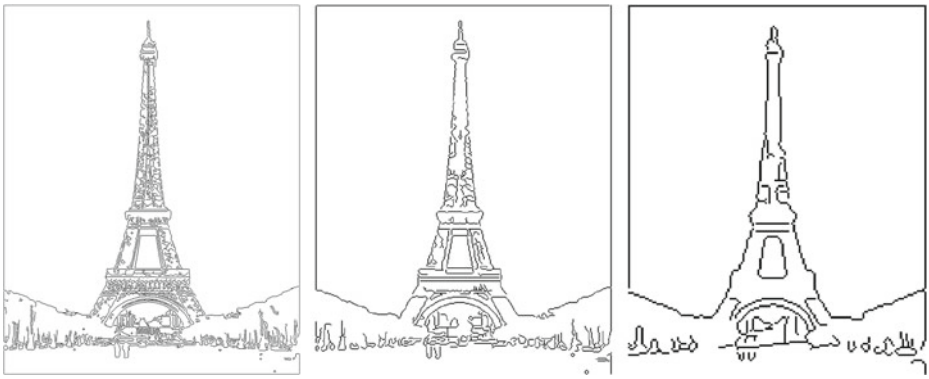


Fig. 4 Edge image produced by a multiscale Canny

A simple edge detection method produces a chaotic edge image. Many of the edge pixels do not provide relevant structural information as we note in Fig. 3. Furthermore, many of them may be a result of noise which may cause degradation in the keyshape detection and consequently in the retrieval effectiveness. To solve this problem, we apply the Canny operator in a multiscale manner. Each scale is computed by the Canny operator applied over a downsampled image. For each scale, an image is downsampled using a factor of 0.5 with respect to the previous scale. In our method, we apply the downsampling process iteratively until the size of the resulting image is less than 200 pixels in one of its dimensions. In our experiments, the downsampling stops after approximately three iterations. We show an example of this process in Fig. 4.

In the same vein, we apply a downsampling approach for the query. However, due to a query already being a sketch we apply, a thinning operation [14] instead of the Canny operator. The result of the this approach is shown in Fig. 5.

After the multiscale stage we keep only the edge map of the third scale, where the noise has been reduced considerably. In addition, small details have been deleted leaving high-scale edges representing the object from a coarser level.

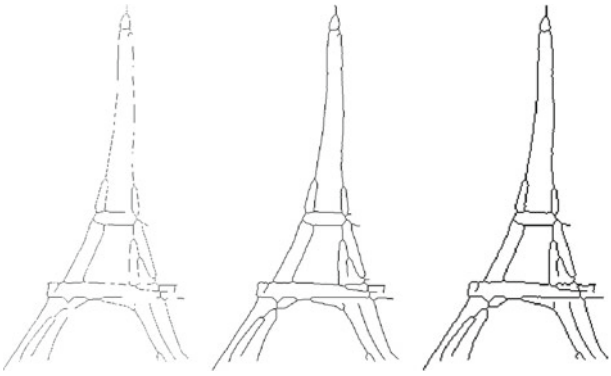


Fig. 5 Edge images produced by a multiscale approach over a sketch

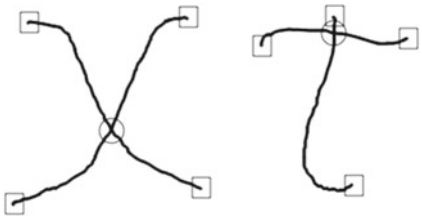


Fig. 6 Branching and terminal points for two sketch images

Having a sketch-like image from both the test image and the query, the next step is to obtain an abstract representation from them that allows us to detect simple shapes in a easy way. A good approach for getting such an abstract representation is to decompose the sketch-like image into strokes.

Definition 3 A stroke is a set of pixels produced when a user is making a line-based hand-drawing between a "pen down" and a "pen up" event.

Considering that the underlying images are produced in an offline environment, we do not have available information about the real strokes. To face this problem, we propose to approximate real strokes by edge links.

3.1.2 Edge links

An edge link is a sequence of edge pixels starting and ending in a branching or terminal point. In Fig. 6, branching points are shown enclosed by circles and terminal points are shown enclosed by squares. We call the branching or terminal points simply *breakpoints*.

A simple algorithm to compute edge links is by tracing neighbor edge pixels starting from a breakpoint until another breakpoint is reached. Having a binary image as input, the output of the algorithm is a set of edge links. The complete description of this algorithm is presented in Algorithm 1. A nice implementation of this algorithm is provided by Kovesi [17].

The algorithm for edge links computation described in Algorithm 1 returns a set of edge links which approximate real strokes. For this reason, henceforth we call these edge links simply "strokes". In Fig. 7, the image of Fig. 6 is depicted with its corresponding strokes which are showed with different colors.

3.1.3 Keyshapes

Detecting simple shapes like circles or ellipses on an image might lead to a time consuming process, so we propose an efficient strategy for detecting six types of *keyshapes*. These *keyshape* classes are: (1) vertical line, (2) horizontal lines, (3) diagonal line (slope $= 1$), (4) diagonal line (slope $= -1$), (5) arc, and (6) ellipse (see Fig. 8). The latter also includes circular shapes. It is important to note that as the number of *keyshapes* increases, the complexity for scaling the method to large databases also increases. For this reason, we decided to keep just six *keyshape* types. In spite of the small number of classes, we still get important structural information from images. Furthermore, we chose to represent a sketch by the six mentioned

Algorithm 1 Edgelink

Require: bim: a binary image

```
E ← edgepixels(bim)
B ← detect_breakpoints(bim)
L ← ∅
for ∀e ∈ E do
    visited(e) ← false
end for
for ∀b ∈ B do
    τ ← ∅
    q ← create a queue
    push(q, b)
    end ← false
    while !empty(q)&!end do
        x ← pop(q)
        visited(x) ← true
        τ ← τ ⋃ x
        if isBreakpoint(x) then
            end ← true
        else
            for ∀v that is a 4-connected neighbor of x do
                if !visited(v) then
                    push(q, v)
                end if
            end for
        end if
    end while
    L ← L ⋃ τ
    visited(b) ← false
    visited(x) ← false
end for
return L
```

primitives because they represent basis shapes from which more complex shapes are formed. We will now describe the complete process for detecting *keyshapes*.

Considering that a stroke may include one or more keyshapes (see Fig. 9), the first step is to divide each stroke S, computed previously, into a set of one or more **stroke pieces** SP_S. To this end, a stroke S is divided with respect to its **inflection points**.

Fig. 7 Strokes approximated by edge links. Different colors indicating different strokes.

Fig. 8 The six clases of keyshapes used in this proposal: *four types of lines*, *arc* and *ellipse*

Definition 4 An inflection point is a point where the local edge pixels around it are distributed significantly in two directions.

To determine whether a stroke point q is actually an inflection point or not, we compute the two eigenvalues (λ_1, λ_2) of the covariance matrix of the points falling in a local region around q. Then, we evaluate the ratio $r = \max(\lambda_1, \lambda_2)/\min(\lambda_1, \lambda_2)$ and proceed to mark q as an inflection point if r is greater than a threshold. We experimentally set this threshold equal to 10.

Trying to evaluate all points of a stroke looking for inflection points is, actually, a time consuming task. To face this problem, instead of applying the inflection point test over all stroke points, we test only a small set of stroke points. We call this set as the *set of maximum deviation points* (SoM). We obtain the SoM by approximating a stroke S, which is defined by a set of points $S = \{p_i, \cdots p_j\}$, by straight lines. To this end, a line L is set between p_i and p_j. Then we look for the point $p_k \in S$ ($k = i \cdots j$), with the maximum distance δ_M to L. We call p_k a *maximum deviation point*. If $\delta_M > \mu$ then we process recursively the resulting substrokes $S_1 = \{p_i, \cdots p_k\}$ and $S_2 = \{p_{k+1}, \cdots, p_j\}$. Finally, all the maximum deviation points found by this method form the SoM. A description of this algorithm is described in Algorithm 2.

We should note that the Algorithm 2 returns the set of maximum deviation points (SoM) together with their corresponding deviation value. In Fig. 9 we show a synthetic example of the stroke decomposition in a set of stroke pieces.

After the stroke decomposition stage, we get a set of stroke pieces $\{sp_1, \cdots, sp_K\}$ for a given stroke S. The next task is to classify each of these stroke pieces as one of the six predefined *keyshape* types. In addition, we could also get the number of straight lines approximating a stroke piece from the SoM. Let N_i be such a number for a stroke piece sp_i. This will be a valuable information for the subsequent steps.

To determine the occurrence of a *keyshape*, we start applying a test which will determine whether sp_i is an ellipse or not. We apply this test only if $N_i \geq 2$. To this end, we use the function *testEllipse*, that will be described afterward, that returns a fitness value indicating how well sp_i is approximated by an ellipse together with the underlying approximated ellipse parameters. If $N_i < 2$ or the ellipse fitness value is less than a threshold, the stroke piece sp_i may be composed by arcs or straight lines. In this way, if $N_i = 1$, a line is detected. Otherwise, we apply a test

Fig. 9 On the *left*, a synthetic example showing inflection points on a stroke. On the *right*, straight lines (*dashed lines*) approximating stroke pieces

Algorithm 2 getSoM

Require: $S = \{p_i, \cdots, p_j\}$
1: **if** $i > j$ **then**
2: **return** $\emptyset$
3: **end if**
4: $L \leftarrow$ a line defined between p_i and p_j.
5: $[k, \delta_M] \leftarrow getMaxDistPoint(L, S)$
6: $SoM \leftarrow \emptyset$
7: **if** $\delta_M > \mu$ **then**
8: $SoM_1 \leftarrow getSoM(\{p_i, \cdots, p_k\})$
9: $SoM_2 \leftarrow getSoM(\{p_{k+1}, \cdots, p_j\})$
10: $SoM \leftarrow SoM_1 \bigcup \{(p_k, \delta_M)\} \bigcup SoM_2$
11: **end if**
12: **return** SoM

for detecting arcs using the function *testArcs*, described later, that returns an error of arc approximation together with the underlying arc parameters. If the error of arc approximation is greater than a threshold we divide sp_i in the point with maximum deviation using the SoM. The resulting two substrokes are tested separately to detect arcs and lines recursively. The algorithms for detecting keyshapes are described in Algorithms 3 and 4.

3.1.4 Detecting ellipses

The *testEllipse* function takes a stroke piece SP and tries to approximate SP by an ellipse. To approximate an ellipse we use the ellipse general equation as in [30].

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + 1 = 0 \tag{1}$$

Algorithm 3 detectKeyshapes

Require: SP: a stroke piece
Require: SoM_{SP}: the set of maximum deviation points with respect to SP, $SoM_{SP} = SoM \bigcap SP$.
$KS \leftarrow \emptyset$
$N_i \leftarrow$ number of straight lines approximating SP.
if $N_i > 2$ **then**
$[t, param] \leftarrow testEllipse(SP)$
if $t > TH_E$ **then**
$KS = \{(\text{“ellipse”}, param)\}$
else
$[lines, arcs] \leftarrow detectArcsLines(SP, SoM_{SP})$
$KS \leftarrow lines \bigcup arcs$
end if
end if
return KS

Algorithm 4 detectArcsLines

Require: $SP = \{p_i, \cdots, p_j\}$: a stroke piece
Require: SoM_{SP}: the set of maximum deviation points with respect to SP, $SoM_{SP} = SoM \bigcap SP$.
  $lines \leftarrow \emptyset$
  $arcs \leftarrow \emptyset$
  $N_i \leftarrow$ number of straight lines approximating SP.
  if $N_i = 1$ **then**
    $lines = \{(\text{“line”}, [p_i, p_j])\}$
  else
    $[e, param] \leftarrow testArc(SP)$
    if $e < TH_A$ **then**
      $arcs = \{(\text{“arc”}, param)\}$
    else
      $k \leftarrow maximumDeviationPoint(SoM_{SP})$
      $SP_1 \leftarrow \{p_i, \cdots p_k\}$
      $SP_2 \leftarrow \{p_{k+1}, \cdots p_j\}$
      $[lines_1, arcs_1] \leftarrow detectArcsLines(SP_1, SP_1 \bigcap SoM_{SP})$
      $[lines_2, arcs_2] \leftarrow detectArcsLines(SP_2, SP_2 \bigcap SoM_{SP})$
      $lines \leftarrow line_1 \bigcup line_2$
      $arcs \leftarrow arcs_1 \bigcup arcs_2$
    end if
  end if
  return $[lines, arcs]$

where the five parameters $(x_c, y_c, r_{\max}, r_{\min}, \theta)$ corresponding to the center of the ellipse (x_c, y_c), the maximum and minimum radii $(r_{\max}, r_{\min})$, and the angle respect to the major radius are obtained as follow:

$$x_c = \frac{BE - CD}{W} \tag{2}$$

$$y_c = \frac{DB - AE}{W} \tag{3}$$

$$r_{\max} = \sqrt{\frac{-2 \cdot det(M)}{W(A + C - R)}} \tag{4}$$

$$r_{\max} = \sqrt{\frac{-2 \cdot det(M)}{W(A + C + R)}} \tag{5}$$

$$\theta = \frac{1}{2} tan^{-1}\left(\frac{2B}{A - C}\right), \tag{6}$$

where

$$W = AC - B^2 \tag{7}$$

$$R = \sqrt{(A-C)^2 + 4B^2} \tag{8}$$

$$M = \begin{pmatrix} A & B & D \\ B & C & E \\ D & E & 1 \end{pmatrix}. \tag{9}$$

To solve the (1) with respect to a stroke piece $SP = \{p_1, \cdots, p_n\}$, we take the following five points: p_1, $p_{\frac{n}{4}}$, $p_{\frac{n}{2}}$, $p_{\frac{3n}{4}}$, p_n. We then validate the estimated ellipse with respect to the pixels on the image. To this end, we use a fitness function defined by Yao et al. [30].

The fitness function takes each point p from the approximated ellipse E and looks for the closer edge pixel on the image. Let d_p be the closer distance from p to any edge pixel on SP, we compute the fitness value as:

$$fitness_E = \frac{\sum_{\forall p \in E} \frac{1}{exp(\gamma d_p)}}{|E|}, \tag{10}$$

where the distance function is the Manhattan distance and γ is a regularization factor that we set to 0.2. The fitness value varies from 0 to 1, where 1 indicates a perfect approximation value while 0 indicates a *null* approximation.

3.1.5 Detecting arcs

Since an arc is actually a segment of a circle, the *testArc* function tries to approximate a circle with five points selected from the underlying stroke piece.

These five points are chosen in the same way as in the case of the *testEllipse* function. The resulting parameters are the circle center (x_c, y_c), and the circle radius r. To evaluate how well a stroke piece SP is approximated by an arc, we compute an approximation error in the following way:

$$error_A = \sum_{p \in SP} |r - dist(p, (x_c, y_c))| \tag{11}$$

where *dist* is the Euclidean distance.

After detecting ellipses, arcs, or straight lines, we represent each detected *keyshape* with a set of parameters. In this way, each *keyshape* is specified as follows:

- **Lines**: $[x_1, y_1, x_2, y_2, L, \iota]$, where (x_1, y_1) is the initial point and (x_2, y_2) is the final point of the detected line. L is the line length. In addition, we use ι to indicate the type of the line (horizontal, vertical, diagonal (slope 1), diagonal (slope -1)).
- **Arcs**: $[x_c, y_c, r]$, where (x_c, y_c) is the center of the estimated circle, and r_c is the corresponding estimated radius.
- **Ellipse**: $[x_c, y_c, r_{\max}, r_{\min}, \phi]$, these are the five parameters of an ellipse, center (x_c, y_c), maximum and minimum radii $(r_{\max}, r_{\min})$, and orientation ϕ.

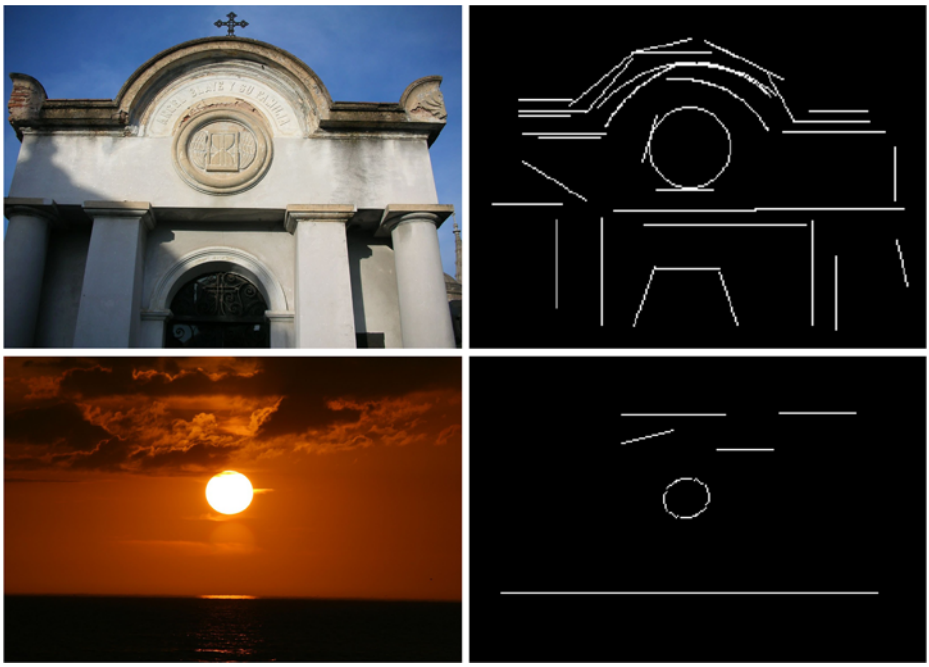


Fig. 10 Test images on the *left* and their *keyshape* images on the *right*.

Finally, we produce a new stroke representation by means of an image called **keyshape image** drawing on it each detected keyshape. The keyshape image might be regarded as a normalized stroke-based representation. Two examples of a **keyshape image** produced by the *keyshape* detection process are shown in Fig. 10.

3.2 Local descriptors

In this section we focus on describing a local region around each keypoint, aiming to represent the distribution of *keyshapes*. We know from Definition 1 that the distribution of the object components is a very important aspect for the structural characterization of the underlying objects. In this stage, we propose two local descriptors for the sketch based image retrieval problem which are characterized by taking into account spatial information of the keyshapes.

The first descriptor is called *Keyshape Angular Spatial Descriptor* (KASD). It divides a local region in an angular way and takes some information from each slice to make a histogram, where the histogram size is the same as the number of slices. The angular partitioning has been used in the computer vision field for the object recognition task [22] and for describing line-base hand-drawings in a global manner [5].

The second descriptor is named *Histogram of Keyshape Orientations* (HKO). It is a SIFT-like descriptor which computes a local histogram of keyshape orientations on a local region around a referent keyshape.

3.2.1 Local region

We define a local squared region with respect to a keyshape (the referent keyshape) in order to compute a local descriptor that characterizes the spatial distribution of the keyshapes around the referent keyshape. The center of the local region coincides with the center position of the underlying keyshape. Thus, let k be a keyshape. If k is a line, the local region is centered on the center of that line. If k is an arc, the local region is centered in the center of the circle containing the arc. In the case of k is an ellipse, the region is centered on the ellipse center.

In order to face with scale variations, we define the region size depending on the keyshape size. Thus, let k be a keyshape and l be the length of the underlying square-like region side, we proceed to compute l as follows.

- If k is a line then $l = length(k) \cdot \eta$.
- If k is an arc then $l = radius(k) \cdot \eta$.
- If k is an ellipse then $l = major\_radius(k) \cdot \eta$.

The symbol η is a constant that we define to be equal to 3. Figure 11 shows a keyshape with the scope of its corresponding local region.

3.2.2 Keyshape Angular Spatial Distribution (KASD)

Let k be a reference keyshape. We divide the local region in angular partitions around k as shown in Fig. 12a. The number of angular regions or slices (N_{SLICES}) is fixed (we suggest using four or eight slices). For each slice, we compute a local histogram that represents the local distribution of the keyshape points with respect to the six keyshape types. Consequently, we obtain a 6-bin histogram for each slice. Then, we concatenate all the local histograms built for each slice to form a $N_{SLICES} \times 6$-size descriptor. Finally, we transform the descriptor into its unit representation.

The spatial distribution of keyshapes is represented by the local histograms spread through the slices in which the local region is partitioned.

3.2.3 Histogram of Keyshape Orientations (HKO)

Let k be a reference keyshape, we divide the local region around k into a 2×2 grid, where each cell of the grid is called a subregion, as shown in Fig. 12b. For each subregion we compute a histogram of orientations. The orientation angle varies between 0 and π, and it is quantized using 8 bins. The keyshape orientation is

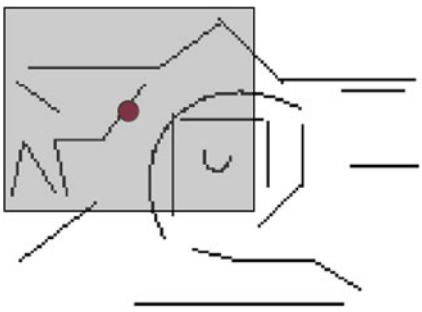


Fig. 11 A local region of around of a diagonal-type keyshape (*that marked with a red filled circle*)

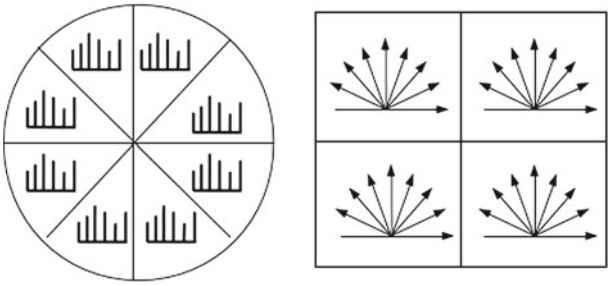


Fig. 12 On the *left*, an angular partitioning descriptor. On the *right*, a histogram of orientations descriptor

computed approximating local gradients using Sobel masks [13]. The local gradients are computed over the keyshape image where many of the outliers have been removed. Therefore, computing orientations over the keyshape image is more robust than computing them over a simple edge map representation. The final descriptor is the concatenation of the four histograms of orientations. We obtain a 32-size descriptor as a result of having four 8-bin histogram for each one of the four subregions. Similar to the above, we take the unit representation of the descriptor.

3.2.4 Combined Descriptor (CD)

To take advantage of both descriptors discussed previously, we propose to use a combined descriptor. This is a descriptor composed of two parts. The first one corresponds to the *Keyshape Angular Partitioning Descriptor* (KASD) and the second to the *Histogram of Keyshape Orientations* (HKO) descriptor. A schematic example of the combined descriptor is depicted in Fig. 13. In this case, we reduce the number of slices for the KASD representation to 4. Thus, the combined descriptor is a 56-size vector, where the first 24 bins correspond to the KASD descriptor and the last 32 bins correspond to the HKO descriptor. As with both descriptors discussed previously, we use the unitary representation of the combined descriptor.

3.3 Matching

As seen in previous sections, both a test image and an input sketch are represented by a set of local descriptors whose size depends on the complexity of the underlying image (for instance, see the images presented in Fig. 10). The next step is to find a way to map descriptors from an input sketch to descriptors computed from a test image in order to get a similarity or dissimilarity score. This score will allow us to rank the test images in an increasing order with respect to the similarity score or in an decreasing order with respect to the dissimilarity score.

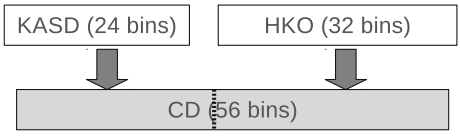


Fig. 13 Scheme of the combined descriptor using a KASD descriptor with 24 bins and a HKO descriptor with 32 bins

Let S be an input sketch and $LD(S)$ be the set of local descriptors of S. We define $LD(S)$ as:

$$LD(S) = \bigcup_{t \in \{v,h,d_1,d_{-1},a,e\}} LD_t(S) \tag{12}$$

where $LD_v(S)$ is the set of local descriptors of vertical lines in S. In the same way, $LD_h(S)$ is set for horizontal lines, $LD_{d_1}(S)$ is set for diagonal lines having slope 1, $LD_{d_{-1}}(S)$ is set for diagonal lines having slope -1, $LD_a(S)$ is set for arcs, and $LD_e(S)$ is set for ellipses.

Further, let I be a test image that will be compared with S. Similarly as the case of S we define $LD(I)$ as:

$$LD(I) = \bigcup_{t \in v,h,d_1,d_{-1},a,e} LD_t(I) \tag{13}$$

The matching process is performed between local descriptors corresponding to the same keyshape type. Since the number of local descriptors is much lower than in the case of having keypoints like those used by the SIFT or the *shape context* approach [1, 19], we could solve an instance of the *bipartite graph problem* using the well known *Hungarian method* [18] between $LD_t(S)$ and $LD_t(I)$, $t = h, v, d_1, d_{-1}, a, e$. The final match results from the union of the partial matches.

Specifically, the Hungarian method solves an instance of the *assignment problem* where, given two sets A,B, the goal is to assign objects of B to objects of A. This produces a one-to-one relationship between A and B. Moreover, an assignment between two objects produces a cost known as the assignment cost. Thus, the objective is to set an appropriate assignment between objects of A and objects of B minimizing the total assignment cost.

Coming back to our case, A is the set of local descriptors of S ($LD(S)$), and B is the set of local descriptors of I ($LD(I)$). The objects are unitary vectors which represent local descriptors. Finally, the cost function we use is the Manhattan distance.

Let $M(S, I)$ be the resulting match between S and I from the previous process, defined as follows:

$$\begin{aligned} M(S, I) = \{&(i_S, j_I, c) | (i_S, j_I, c) \text{ is a match} \\ &\text{whose cost is } c, \wedge i_S \in LD(S) \wedge \\ &j_I \in LD(I)\}, \end{aligned} \tag{14}$$

we define the following properties on M:

- $|M(S, I))|$: The cardinality of M, i.e the number if matches between S and I.
- $A(M(S, I))$: The average match cost defined as:

$$A(S, I) = \sum_{\forall (i,j,c) \in M} \frac{c}{|M(S, I)|}. \tag{15}$$

  In the case that no matches occur, $A(S, I) = \max(LD(S), LD(I))$.
- $U(M(S, I))$: The number of unmatched descriptors. It is defined as follows:

$$U(S, I) = \max(LD(S), LD(I)) - |M(S, I)| \tag{16}$$

3.3.1 Match filtering

The matching is followed by a filtering step that aims to discard matches discordant with a pose estimation. To this end, we apply a vote based approach that takes into account spatial information from the corresponding matched keyshapes in order to estimate a geometric transformation. This transformation considers the scale and location parameters of the computed keyshapes. In this way, each match vote for a certain transformation, the predominant transformation will correspond to the pose estimation. Only the matches that voted for such a pose are held, the remaining matches are discarded. In Algorithm 5 we show how to filter a set of matches.

Algorithm 5 Match Filtering

Require: M: the set of matches.
Require: K_S: The set of keyshapes of the input sketch.
Require: K_I: The set of keyshapes of the test image.
for $\forall m = (i_S, j_I, c) \in M$ **do**
$k_S \leftarrow$ the keyshape of i_S
$k_I \leftarrow$ the keyshape of j_I
$(x_S, y_S) \leftarrow center\_of(k_S)$
$(x_I, y_I) \leftarrow center\_of(k_I)$
$s_S \leftarrow scale(k_S)$
$s_I \leftarrow scale(k_I)$
$d_x = x_I - x_S \frac{s_I}{s_S}$
$d_y = y_I - y_S \frac{s_I}{s_S}$
$q_x = quantize(d_x)$
$q_y = quantize(d_y)$
$matches(q_x, q_y) \leftarrow matches(q_x, q_y) \bigcup m$

end for
the_matches $\leftarrow$ the set of matches in *matches* with the major number of elements.
return *the_matches*

In Algorithm 5, $scale(k)$ gives the scale of a keyshape k. The scale for lines is simply the length of the underlying line, the scale for arcs is the associated radius, and the scale for ellipses is the corresponding major radius. In addition, for the quantization step we divide both the x-dimension and the y-dimension by a factor 3.

3.3.2 Dissimilarity score

After the match filtering step we need to get a score value that represents the similarity or dissimilarity between both images. To this end, we define three dissimilarity functions based on the average match cost, the number of matches, and the number of unmatched descriptors.

1. $D_1(S, I) = A(S, I) + \beta \cdot U(S, I)$.
2. $D_2(S, I) = 1/|M(S, I)|$.
3. $D_3(S, I) = A(S, I)/|M(S, I)|$.

The dissimilarity function can be regarded as a cost function. Thus, D_1 and D_3 are based on the average match cost $A(\cdot, \cdot)$. Furthermore, the number of unmatched descriptors $U(\cdot, \cdot)$ also can express a cost function. Thus, as the value of $U(\cdot, \cdot)$

increases, the resemblance level between the comparing images decreases. In contrast, as the number of matches $|M(\cdot,\cdot)|$ increases, the resemblance level between both images also increases. Therefore, we also use the number of matches in D_2 and D_3, but inversely. Furthermore, in the case of D_1 we use a factor $\beta = 0.15$ with respect to the number of unmatched descriptors to avoid any possible bias effect due to the different ranges of the two involved functions.

To exploit the benefits of all dissimilarity functions defined above, we compute different rankings combining local descriptor with dissimilarity functions. Thus, we define $r^S_{(D,F)}$ as the resulting ranking for an input sketch S, where D indicates one of the local descriptors (KASD, HKO, or CD) and F is one of the dissimilarity functions (D_1, D_2, or D_3) that allows us to sort the test images.

Let $rank(r, \Gamma)$ be a function that provides the position where a test image Γ appears in the ranking r. For computing the final rank, we need to determine a new score (the final score) for each test image. This score is computed as follows:

$$final\_score(\Gamma) = \sum_{\forall r} rank(r, \Gamma). \tag{17}$$

The final ranking then is formed in an increasing order with respect to the new scores. Images with low scores must appear first in the final ranking. Therefore, if an image appears in the top of all rankings, it also will appear in the top of the combined ranking.

In order to get the rankings, we propose to use the followings:

- $r_{(CD,D_2)}$: Ranking produced by the combined descriptor and the dissimilarity function D_2.
- $r_{(KASD_4,D_3)}$: Ranking produced by the KASD descriptor using 4 partitions and the dissimilarity function D_3.
- $r_{(KASD_8,D_1)}$: Ranking produced by the KASD descriptor using 8 partitions and the dissimilarity function D_1.

3.4 Computational complexity

In this section, we discuss the computation complexity of our approach. For the sake of clarity, we divide the discussion of the computational complexity of our algorithms in three groups. First, we discuss the complexity of the detection keyshape stage. Second, we discuss the complexity of computing the proposed descriptors. Finally, we discuss the complexity of comparing a test image with an input sketch.

3.4.1 Keyshape detection

This stage involve many algorithms including pre-processing tasks, *edgelink* detection, inflection point detection and the keyshape classification.

- Pre-processing: This task consists in analyzing each pixel of an image to obtain an edge map representation. The complexity of this task is $O(N)$ where N is the number of pixels of the underlaying image.
- EdgeLink: In this stage, we trace the edge map pixels to determine potential strokes. Therefore, the complexity of this algorithm is $O(E)$, where E is the number of edge pixels.

- Set of Maximum Deviation Points: To get the *Set of Maximum deviation* (SoM), the algorithm receives a set of strokes and, by approximating straight lines, determines points of maximum deviation with respect to those lines. Tho this end, the algorithm has to evaluate a number of points proportional to the size of E. Therefore, the complexity of $getSoM$ is $O(E)$.
- Inflection Points: To determine inflection points the algorithm evaluates each maximum deviation point together with a local region of 25×25 pixels. As the size of SoM is less than the size of E, the complexity of this algorithm is also bounded by $O(E)$.
- Detection of Keyshapes: In this stage, each stroke piece is evaluated to determine what keyshape the underlying stroke piece corresponds to. However the algorithm for classifying arcs and lines may have a cost proportional to the number of straight lines that approximate a stroke piece. Therefore, this stage has a complexity of $O(NL)$, where NL is the number of approximating straight lines, which is less that the size of E.

Therefore, the complexity of detecting keyshapes is $O(N + E)$. However, since $N > E$, the complexity results to be linear with respect to the image size.

3.4.2 Keyshape descriptors

We have presented two descriptors. The first one, KASD, processes an image in time $O(K^2)$, where K is the number of keyshapes. The second approach, HKO, runs in time $O(K \times W)$, where W is the size of the local regions around a descriptor is computed.

3.4.3 Keyshape matching

We use the Hungarian method to find a set of matches between two sets of keyshapes. This algorithm runs in cubic time with respect to the size of the input [28]. As our input is a set of keyshapes, the time of the algorithms runs in $O(K^3)$, where K is the number of keyshapes . In practical issues, the average number of keyshapes computed in our data set is approximately 30, which requires low computational cost in terms of matching. Experimentally, the reported time for comparing two sets of keyshapes is approximately 8 ms.

In conclusion, the total cost for comparing two images is $O(N + K \times W + K^3)$, where N is the number of pixels in the image, K is the number of keyshapes and W is the size of the local region used for computing local descriptors.

4 Experimental evaluation

4.1 The benchmark

Due to sketch-based image retrieval being a young research area, there are a few benchmarks for comparing SBIR methods. In this regard, Eitz et al. [11] conducted an interesting systematic work to propose a benchmark in this area. We chose this benchmark because this is the only benchmark that takes into account the user's opinion about the relevance of an image with respect to an input sketch. This fact

turns very important as the main goal of a retrieval system is, precisely, to calculate a ranking very close to what the user is expecting to get.

This benchmark consists of 31 query sketches, each one associated with a set of 40 test images. An example of two query sketches and seven of their associated test images in the data set are shown in Fig. 14.

Furthermore, each test image has been ranked by people using a 7-point Likert scale [11] with 1 representing the best rank to 7 representing the worst rank. This provides the baseline ranking which is called the *user ranking*.

We proceed to compare the ranking of the proposal method against the user ranking. To this end, Eitz el al. propose to use the *Kendall's correlation* τ that determines how similar two rankings are. The correlation coefficient τ may vary from 1 to -1, with -1 indicating that one ranking is the reverse of the other, 0 indicating

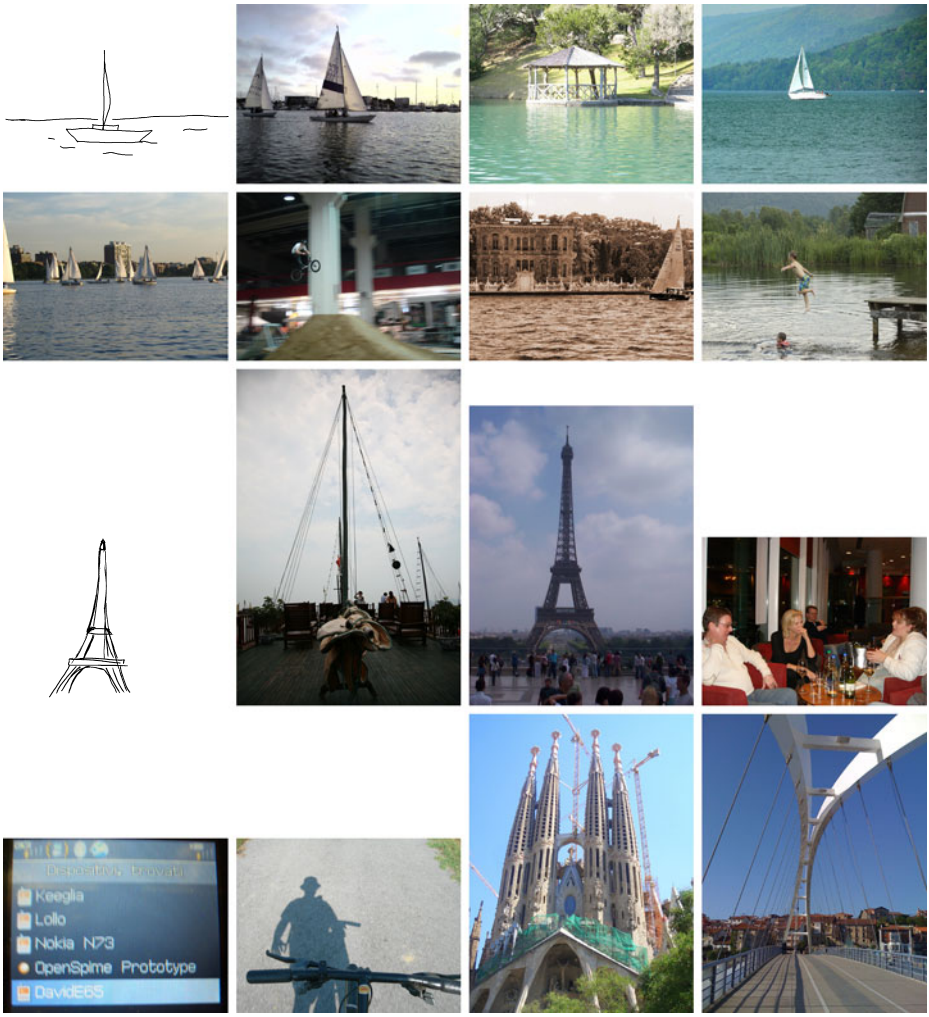


Fig. 14 Two query sketches with seven associated images.

independence between the rankings, and 1 indicating that the two rankings have the same order. Therefore, as we achieve a correlation value near 1, our proposal will be better, indicating that the resulting ranking is very similar to that expected for the users.

Considering that both the user ranking and the ranking of a proposal method may include tied scores, a variation of the *Kendall's correlation* is used. This variation is denoted by τ_b and it is defined as:

$$\tau_b = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}, \tag{18}$$

where

- n_c = Number of concordant pairs.
- n_d = Number of discordant pairs.
- $n_0 = n(n-1)/2$.
- $n_1 = \sum_{i=1}^{t} t_i(t_i - 1)/2$.
- $n_2 = \sum_{i=1}^{u} [u_i(u_i - 1)/2$.
- t_i = Number of tied values in the ith group of ties for the user ranking.
- u_i = Number of tied values in the ith group of ties for the proposed ranking.
- t = Number of groups of ties for the user ranking.
- u = Number of groups of ties for the proposed ranking.

4.2 Involved parameters

In order to make our approach be repeatable, we present in Table 1 the values of the parameters used during the experimental evaluation.

4.3 Discussion of the results

In the experiments conducted by Eitz et al. [11], the best correlation value they achieved was 0.277. This result was obtained using a Bag of Features methodology with a variant of the histogram of gradient descriptor. The approach uses a 1000-size codebook, that means that the codebook is formed with 1000 codewords. In addition, a correlation value under 0.18 is reported for the Shape Context method [1].

Our results show a slight increment in the effectiveness of the retrieval process without requiring a tedious learning stage. We achieve a correlation value of 0.289. A graphic showing the difference correlation value for different SBIR methods is depicted in Fig. 15. Considering that our proposal is based on the structural feature of a sketch representation, which has not been taked into account for current methods, our method is potentially adequate for being combined with a leading method to increase the retrieval effectiveness.

Table 1 Values of the parameters involved in our approach used during the evaluation process

Parameter	Description	Value
μ	Threshold for detecting points of maximum deviation	5
TH_E	Threshold for detecting ellipses	0.8
TH_A	Threshold for detecting arcs	0.7
η	Factor for defining a local region	3

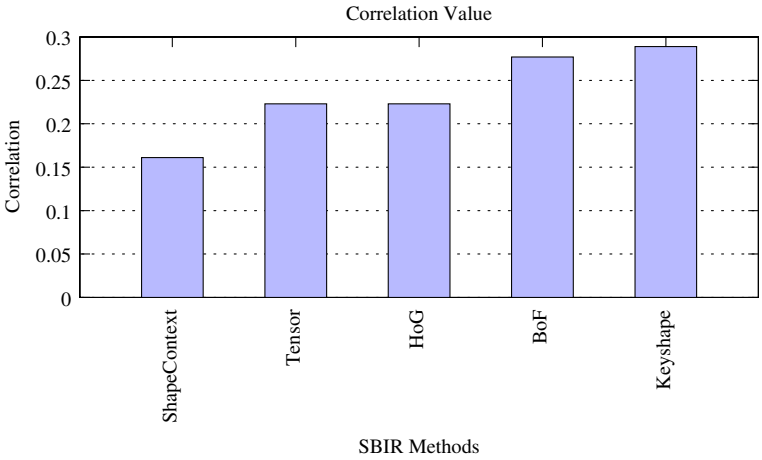


Fig. 15 Kendall's correlations for SBIR methods. Our proposal (*the last bar on the right*) outperforms that of the state of the art. Our keyshape based approach achieves a correlation value of 0.289 which outperforms the correlation achieved by the BoF approach proposed by Eitz et al. (0.277)

To show the goodness of combining our method with a current leading method, we show, in this document, the results of combining our proposal with the BoF approach proposed by Eitz et al. [11]. For combination, we follow the same approach we use to combine our two proposed descriptors. In Fig. 16 we show that the combined method (Keyshape+BoF) allows us to achieve a correlation value of 0.337 which means an increase of almost 22 % with respect to the correlation value achieved by the BoF approach. To validate these results, we run a statistical test (T-Test) comparing the results produced by the BoF approach and the results achieved by

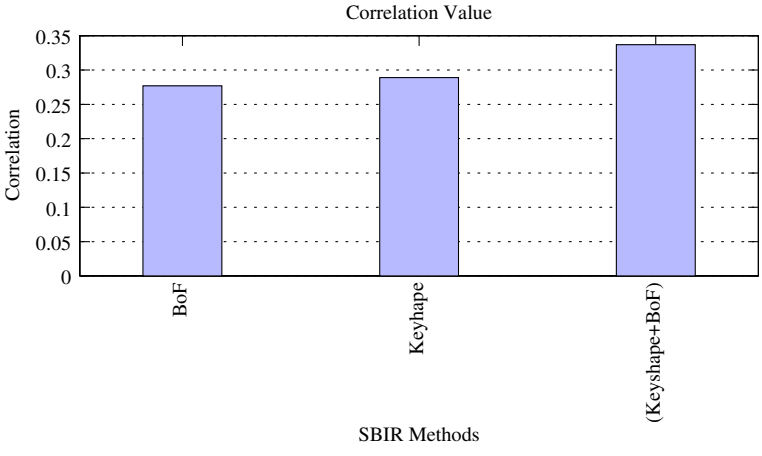


Fig. 16 Kendall's correlations for the approaches: BoF (0.277), *Keyshape*-based (0.289), and the Keyshape+BoF (0.337)

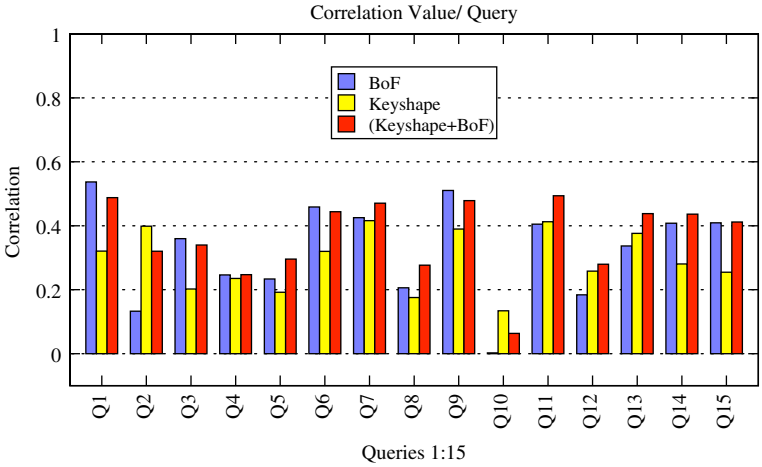


Fig. 17 Correlation values for the first 15 queries using the BoF, *Keyshape*-based and *Keyshape*+BoF approaches

our combined proposal. The statistical test gave a *p_value* of 0.1 % which indicates that our combined method improves significantly the BoF approach.

Furthermore, we present in Figs. 17 and 18 the correlation value achieved for each sketch using the BoF approach, our keyshape-based proposal and the combined method. We note that although our approach gains a significant improvement for some queries (for instance, see query 25 and 22), the overall improvement is not significant enough. However, when we analyze the correlation achieved by the combined method, we see that it achieves an improvement in 24 queries with respect to the BoF results. Additionally, the overall effectiveness is improved in almost 22 %.

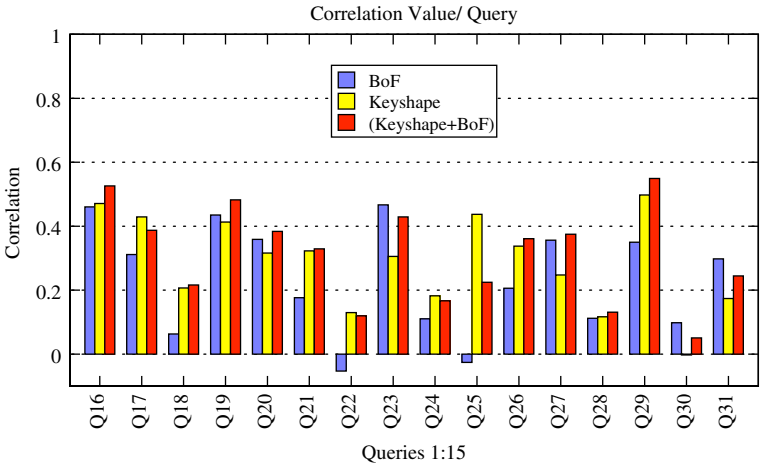


Fig. 18 Correlation values for the last 16 queries using the BoF, *Keyshape*-based and *Keyshape*+BoF approaches

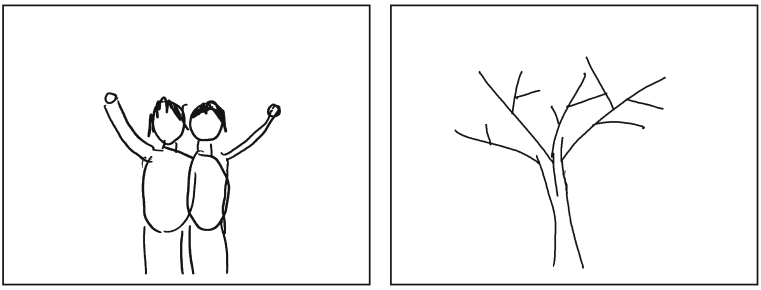
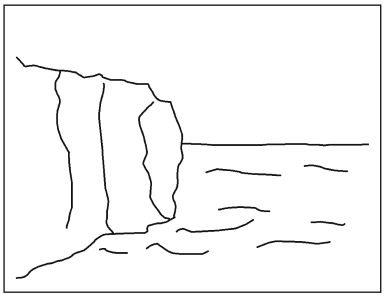


Fig. 19 Sketch queries for what our method achieves significant improvement with respect to the BoF approach. Q2 appears on the *left* and Q25 appears on the *right*

From our results showed in Figs. 17 and 18 we can also note that our method achieves impressive improvement for queries Q2 and Q25 (see Fig. 19) with respect to the correlation achieved by the BoF approach (BoF achieves negative correlation). In the case of Q2, our keyshape based approach achieves a correlation value of 0.3984 which is much better that the correlation (0.1326) achieved by the Eitz's approach. In the case of Q25, our keyshape approach achieves a correlation of 0.4373 which is again higher than the correlation value achieved by the Eitz' approach, that in this case is negative (−0.0258). A possible reason for this fact is that these queries can be represented by a set of very discriminative keyshapes. In particular, Q2 can be represented by a set of arcs and ellipses, while Q25 can be represented by a family of straight lines. In this sense, the keyshape based descriptors seems to be very appropriated to represent them.

It is also important to note that some images may not be good represented by keyshapes. In particular, we note that Q30 (see Fig. 20) is the query with the lowest correlation achieved by our approach (−0.0026). This poor result may be explained from the viewpoint of the test images related with this sketch. Indeed, although the sketch may be appropriately represented by a set of arcs or straight lines, the images related with this sketch seem to be badly represented by our approach. After analysing these images, we realized that these are characterized by being cluttered image which may also justify the poor performance obtained by BoF approach (0.0979). This shows that a cluttered image is not only a problem for our approach but also for current approaches. Therefore, dealing with cluttered images represents a challenging problem that we have to address in the area of sketch based image

Fig. 20 A query for what our method achieves the lowest correlation (−0.0026)



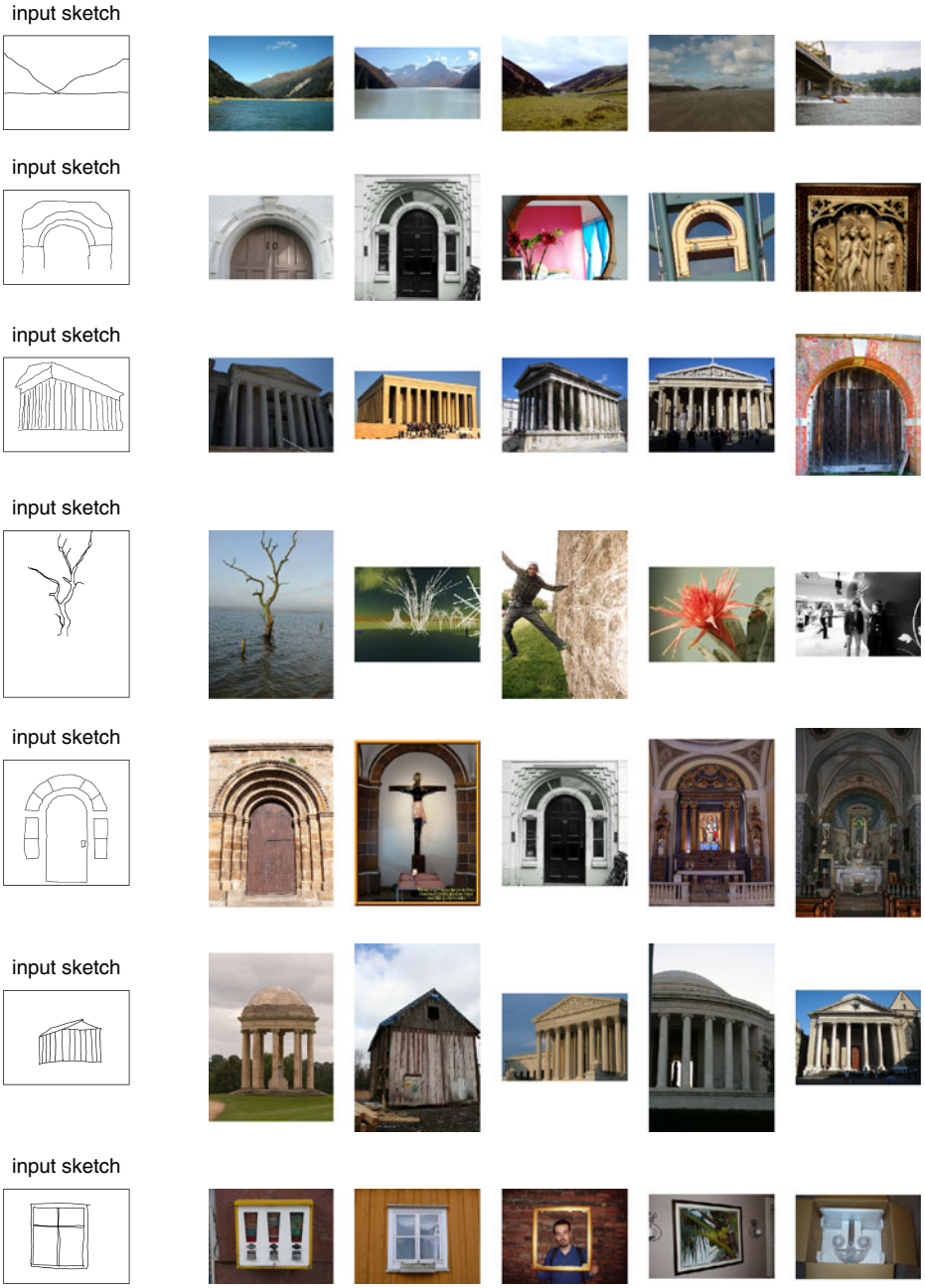


Fig. 21 Example of SBIR using our proposal after retrieving the first five images from the test database. The *first column* shows the input sketch (the query) and the *next five columns* show the first five retrieved images, respectively

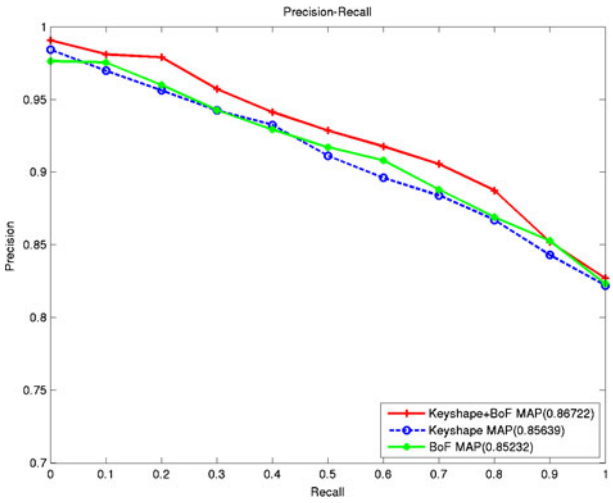


Fig. 22 Precision-Recall graphic comparing the BoF, *Keyshape*-based and *Keyshape*+BoF approaches

retrieval. Additionally, to show the performance of our proposal, we present in Fig. 21 the top five retrieved images for seven input sketches.

Finally, although the benchmark used in this work is focused in evaluating the correlation of the resulting rankings, we also evaluate the precision and recall after retrieving the correspondent images for each sketch. In the Fig. 22 we show the precision-recall graphic comparing the BoF, *Keyshape*-based and *Keyshape*+BoF approaches. This evaluation shows that the performance of the keyshape based proposal and the BoF approach are similar. However, the combination of these two approaches allow us to increase the retrieval effectiveness achieving a MAP of 0.87. This again shows that our keyshape based approach exploits different features from those exploited by the BoF approach which allow us to get a feasible combination.

5 Conclusions

In this paper we have presented a novel local method for sketch based image retrieval. Our method is based on detecting simple shapes called *keyshapes*. This allows us to get structural representation of images leading to an improvement of retrieval effectiveness. Our proposal improve the results of the state of the art methods achieving a correlation value of 0.289.

Furthermore, we analyze a combined proposal exploiting the results of the BoF approach and the results of our *keyshape* based approach showing that our method is complementary to the BoF approach. This fact is reflected by the results achieved by the combination of both methods. The results shown that using a combined method allows us to increase the retrieval effectiveness in almost 22 %. Additionally, we shown that this result is significantly better than that of the state-of-the-art methods.

In addition, we have presented an efficient method for detecting simple shapes on an image. This strategy could be applied for other applications requiring a reduction of the image complexity.

Of course, sketch based image retrieval is still a challenging task. In this vein, our current work is focused on defining a more efficient metric for comparing *keyshapes*. In addition, we are working on evaluating our work in large scale dataset as well as extending our approach to the sketch based 3D models retrieval.

Acknowledgement We thank CONICYT-CHILE for supporting this work through the doctoral scholarship number 63080026.

References

1. Belongie S, Malik J, Puzicha J (2002) Shape matching and object recognition using shape contexts. IEEE Trans Pattern Anal Mach Intell 24:509–522
2. Borgefors G (1988) Hierarchical chamfer matching: a parametric edge matching algorithm. IEEE Trans Pattern Anal Mach Intell 10(6):849–865
3. Canny J (1986) A computational approach to edge detection. IEEE Trans Pattern Anal Mach Intell 8(6):679–698
4. Cao Y, Wang C, Zhang L, Zhang L (2011) Edgel index for large-scale sketch-based image search. In: Proceedings of the 2011 IEEE conference on computer vision and pattern recognition. IEEE Computer Society, pp 761–768
5. Chalechale A, Naghdy G, Mertins A (2005) Sketch-based image matching using angular partitioning. IEEE Trans Syst Man Cybern Syst Hum 35(1):28–41
6. Chen T, Cheng MM, Tan P, Shamir A, Hu SM (2009) Sketch2photo: internet image montage. ACM Trans Graph 28(5):124:1–124:10
7. Dalal N, Triggs B (2005) Histograms of oriented gradients for human detection. In: Proceedings of the 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05), vol 1. IEEE Computer Society, pp 886–893
8. Del Bimbo A, Pala P (1997) Visual image retrieval by elastic matching of user sketches. IEEE Trans Pattern Anal Mach Intell 19(2):121–132
9. Eitz M, Hildebrand K, Boubekeur T, Alexa M (2009) A descriptor for large scale image retrieval based on sketched feature lines. In: Proc. of the 6th eurographics symposium on sketch-based interfaces and modeling, pp 29–36
10. Eitz M, Hildebrand K, Boubekeur T, Alexa M (2009) Photosketch: a sketch based image query and compositing system. In: SIGGRAPH 2009: Talks, SIGGRAPH '09, pp 60:1–60:1
11. Eitz M, Hildebrand K, Boubekeur T, Alexa M (2011) Sketch-based image retrieval: benchmark and bag-of-features descriptors. IEEE Trans Vis Comput Graph 17(11):1624–1636
12. Funkhouser T, Min P, Kazhdan M, Chen J, Halderman A, Dobkin D, Jacobs D (2003) A search engine for 3d models. ACM Trans Graph 22(1):83–105
13. Gonzalez R, Woods R (2008) Digital image processing, 3rd edn. Pearson Prentice Hall, New Jersey
14. Guo Z, Hall RW (1989) Parallel thinning with two-subiteration algorithms. Commun ACM 32(3):359–373
15. Hu R, Barnard M, Collomosse J (2010) Gradient field descriptor for sketch based retrieval and localization. In: 17th IEEE International conference on image processing (ICIP), 2010, pp 1025–1028
16. Hu R, Wang T, Collomosse J (2011) A bag-of-regions approach to sketch-based image retrieval. In: 18th IEEE International conference on image processing (ICIP), pp 3661–3664
17. Kovesi PD (2000) MATLAB and octave functions for computer vision and image processing. School of Computer Science & Software Engineering, The University of Western Australia. Available from: http://www.csse.uwa.edu.au/~pk/research/matlabfns/
18. Kuhn HW (2010) The hungarian method for the assignment problem. In: 50 Years of integer programming 1958–2008, pp 29–47
19. Lowe DG (2004) Distinctive image features from scale-invariant keypoints. Int J Comput Vis 60:91–110
20. Martin D, Fowlkes C, Tal D, Malik J (2001) A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In: 8th International conference on computer vision, vol 2, pp 416–423

21. Martínez JM (2002) MPEG-7: overview of MPEG-7 description tools, part 2. IEEE Multimedia 9(3):83–93
22. Mikolajczyk K, Schmid C (2005) A performance evaluation of local descriptors. IEEE Trans Pattern Anal Mach Intell 27(10):1615–1630
23. Saavedra J, Bustos B (2010) An improved histogram of edge local orientations for sketch-based image retrieval. In: 32nd annual symposium of the German association for pattern recognition (DAGM). no. 6376 in lecture notes in computer science. Springer-Verlag, pp 432–441
24. Saavedra JM, Bustos B, Scherer M, Schreck T (2011) Stela: sketch-based 3d model retrieval using a structure-based local approach. In: Proceedings of the 1st ACM International Conference on Multimedia Retrieval, ICMR '11. ACM, pp 26:1–26:8
25. Stenger B, Thayananthan A, Torr PHS, Cipolla R (2006) Model-based hand tracking using a hierarchical bayesian filter. IEEE Trans Pattern Anal Mach Intell 28(9):1372–1384
26. Sun Won C, Kwon Park D, Park SJ (2002) Efficient use of MPEG-7 edge histogram descriptor. ETRI 24:23–30
27. Tuytelaars T, Mikolajczyk K (2008) Local invariant feature detectors: a survey. Now Publishers Inc., Hanover, MA
28. Wong J (1979) A new implementation of an algorithm for the optimal assignment problem: an improved version of munkres' algorithm. BIT 19:418–424
29. Xu L, Oja E, Kultanen P (1990) A new curve detection method: Randomized Hough Transform (RHT). Pattern Recogn Lett 11:331–338
30. Yao J, Kharma N, Grogono P (2005) A multi-population genetic algorithm for robust and fast ellipse detection. Pattern Anal Applic 8:149–162

Jose M. Saavedra obtained the PhD degree in Computer Science from the Department of Computer Science, University of Chile (2013). In addition, he also received a Master degree in Computer Science (2006) from National University of Trujillo, Peru. Currently, he works as a computer vision researcher at ORAND S.A (Chile). His current research is focused on content based image retrieval, objet recognition, multimedia retrieval and handwriting recognition.

Benjamin Bustos received the doctoral degree in natural sciences from the University of Konstanz, Germany, in 2006. He is an associate professor in the Department of Computer Science, University of Chile, where he is heading the PRISMA Research Group. His research interests include similarity search, multimedia information retrieval, 3D object retrieval, (non)-metric indexing, and pattern recognition.