Efficient Computation of Histograms on Densely Overlapped Polygonal Regions

Yuting Zhang[a], Yueming Wang[b,*], Gang Pan[a], Zhaohui Wu[a]

[a]*Department of Computer Science, Zhejiang University, Hangzhou 310027, P. R. China*
[b]*Qiushi Academy for Advanced Studies, Zhejiang University, Hangzhou 310027, P. R. China*

Abstract

This paper proposes a novel algorithm to efficiently compute the histograms in densely overlapped polygonal regions. An incremental scheme is used to reduce the computational complexity. By this scheme, only a few entries in an existing histogram need to be updated to obtain a new histogram. The updating procedure makes use of a few histograms attached to the polygon's edges, which can be efficiently pre-computed in a similar incremental manner. Thus, the overall process can achieve higher computational efficiency. Further, we extend our method to efficiently evaluate objective functions on the histograms in polygonal regions. The experiments on natural images demonstrate the high efficiency of our method.

Keywords: polygon, efficient computation, histogram, incremental computation

1. Introduction

Histogram is one of the most frequently used techniques to encode image features, e.g., histograms of oriented gradient (HoG) [1], histograms of color intensity [2], local binary pattern (LBP) [3], scale-invariant feature transform (SIFT) [4], and bag of visual words [5]. These features are used for detection [1, 6], recognition [7], image classification [8], tracking [9], flow estimation [10], image retrieval [11], face liveliness detection [12], and many other vision tasks. In some of these applications, histograms are computed in the neighborhood regions around keypoints [4] that are sparsely located on images. In other cases, histograms are computed in densely overlapped regions [1, 6] that are organized in regular grids. The keypoint-based paradigm of feature extraction is widely used for images [4] and even for time series [13] due to its high efficiency that attributes to the sparsity of the keypoints. In contrast, the grid-based paradigm can produce richer image features with a much higher computational cost. Generally, the feature richness can significantly benefit the solution of vision tasks [10], such as improving the accuracy rate of the detection and recognition applications. To make practical use of the richer image features produced by the grid-based paradigm, improving the efficiency of histogram computation in densely overlapped regions is very important.

There are several algorithms in the literature to efficiently compute the image histograms in densely overlapped regions. Porikli *et al*. [14] applied the integral image [15] to histogram computation. When the number of histogram bins is much smaller than the number of pixels in the region, better efficiency was obtained with a high memory cost. Otherwise, it is even worse than the simplest method. Sizintsev *et al*. [16] utilized the overlap between regions to reduce redundant computation. In [17], Wei *et al*. presented a similar idea and extend it to the function evaluation on histograms, making the evaluation more scalable to the bin number and the region size, which are usually large in computer vision problems.

Most existing algorithms compute histograms in rectangular regions. However, many real-world objects show diverse shapes and poses. For example, the whole shape of an airplane is irregular. The wings are triangular, the engines are oval, and the windows appear to be parallelograms in most poses. Thus, the image features often include noisy background or lose necessary foreground information if we merely use rectangular regions. Hence, features extracted in rectangular regions are insufficient to describe images well in many circumstances.

In order to obtain more discriminative features, it is necessary to compute the histograms in non-rectangular regions. Pham *et al*. [18] used the triangular integral image to efficiently compute polygonal Haar-like features, and demonstrated that features in suitable non-rectangular local regions can better describe the visual characteristics of images. Compared with the Haar-like feature, the histogram in the region with a suitable shape can be more powerful and discriminative in describing visual information. The triangular integral image can be straightforwardly generalized for histogram computation, nevertheless it does not scale well on the histogram bin number as [14] did on rectangular regions. So far, to efficiently compute histograms in non-rectangular regions is still an unsolved problem, and there is no work to integrate efficient function evaluation into histogram computation for non-rectangular regions.

This paper proposes a novel algorithm to address the problem. We focus on the case of computing histograms in polygonal regions, and consider the typical situation in object detection and image retrieval as well as in many cases of other tasks, in which each polygon is required to slide along the rows and columns of images, thus generating a large number of densely overlapped polygonal regions. Given two overlapped regions, if

[*]Corresponding author. Email: ymingwang@gmail.com

the histogram in one region is computed, we update it according to two regions' non-overlapped areas to obtain the histogram in the other one. The non-overlapped areas is broken into smaller ones. The histogram in each smaller area can be pre-computed efficiently. In addition, we extend this incremental scheme to the problem of evaluating objective functions on the histograms in polygonal regions.

Wei *et al.*'s method [17] for rectangular regions can be taken as a special case of ours. Compared with it, our method is designed with more decent efforts on constructing the incremental computation strategy, handling the computational procedure, and devising more efficient data structure. The proposed method has no strong connection with Pham *et al.*'s triangular integral image [18]. However, their idea of decomposing computation according to the edges of a polygon gives us inspiration.

We compare the proposed algorithm with several other methods, including the most straightforward method, the generalization of the triangular integral image, and a simplified variation of our method. The experimental results show that the proposed algorithm is computationally more efficient.

2. Basic Idea

Given a $W \times H$ index bitmap with B indices, we represent it as a function $A(x, y) \in \{1, 2, \ldots, B\}$ for $x \in \{1, 2, \ldots, W\}$ and $y \in \{1, 2, \ldots, H\}$. By locating the left-top corner of the index bitmap at the origin of $\mathbb{R}^2$ (y axis pointing down), A is generalized to the continuous domain as

$$a(x, y) = \begin{cases} A(\lceil x \rceil, \lceil y \rceil), & 0 < x \le W,\ 0 < y \le H, \\ 0, & \text{otherwise}, \end{cases} \tag{1}$$

where $(x, y) \in \mathbb{R}^2$, $\lceil \cdot \rceil$ is the ceiling function, and the pixel (x, y) on A is the unit square region $(x-1, x] \times (y-1, y]$ on $\mathbb{R}^2$. Given a region $\Phi \in \mathbb{R}^2$, we denote the histogram of a in it as

$$\mathbf{c}(\Phi) = (c_1(\Phi), c_2(\Phi), \ldots, c_B(\Phi)), \tag{2}$$

where,

$$c_b(\Phi) = \iint_\Phi \iota_b(a(x, y))\mathrm{d}x\mathrm{d}y, \tag{3}$$

$\iota_b(t) = 1$ if $t = b$, and $\iota_b(t) = 0$ if $t \neq b$.

For two regions Φ and Φ', the following equations holds,

$$\mathbf{c}(\Phi') = \mathbf{c}(\Phi) + \delta\mathbf{c}(\Phi', \Phi), \tag{4}$$

where,

$$\delta\mathbf{c}(\Phi', \Phi) = \mathbf{c}(\Phi' \backslash \Phi) - \mathbf{c}(\Phi \backslash \Phi') \tag{5}$$

is the *overall residual* histogram, and "\" denotes set subtraction. As shown in Fig. 1a, if the region Φ' is obtained by moving Φ a little rightward, Φ and Φ' have much overlap. $\Phi' \backslash \Phi$ is the newly emerged region in Φ' which is called the *positive residual* in this paper. Similarly, $\Phi \backslash \Phi'$ is the *negative residual*. According to (4), if $\mathbf{c}(\Phi)$ is known, $\mathbf{c}(\Phi')$ can be computed from $\mathbf{c}(\Phi)$ by adding only the non-zero entries of $\delta\mathbf{c}(\Phi', \Phi)$. Indeed, the number of these non-zero entries is usually very small compared with the bin number of $\delta\mathbf{c}(\Phi', \Phi)$. This is true if the number of pixels inside $\Phi' \backslash \Phi$ and $\Phi \backslash \Phi'$ is smaller than the bin number. More importantly, we observed that nearby pixels in index bitmaps used for visual feature extractions usually have the same value with high probability. This phenomenon attributes to the local smoothness of the natural images, from which the index bitmap is obtained. As a result, no matter how large $\Phi' \backslash \Phi$ and $\Phi \backslash \Phi'$ are, $\delta\mathbf{c}(\Phi', \Phi)$ is usually very sparse when the movement between Φ and Φ' are not too large. Hence, the incremental computation scheme in (4) helps to improve the efficiency in computing $\mathbf{c}(\Phi')$ and further improve the overall computational efficiency of all histograms on an index bitmap.

It is worth pointing out that, in many cases (e.g. the computation of HoG [1]), different pixels on the index bitmap are associated with different weights. Let $\gamma(x, y)$ be the bitmap of pixel-wise weights. (3) then becomes $c_b(\Phi) = \iint_\Phi \gamma(x, y) d_b(a(x, y))\mathrm{d}x\mathrm{d}y$. In this case, (4) still holds, so our algorithm also works with index bitmaps of weighted pixels.

The idea of incremental filtering has a long history. It appeared early in [19], and was surveyed by [16] for histogram computation in rectangular regions. In [17], the importance of local smoothness on incremental histogram computation was demonstrated. Here, we extend these idea to the case of non-rectangular regions.

3. Incremental Histogram Computation with Residual Segmentation

The computation efficiency of (4) is determined by both the nature of index bitmaps and how efficiently the overall residual histogram $\delta\mathbf{c}(\Phi', \Phi)$ can be computed for overlapped regions Φ and Φ'. In this section, we present an efficient algorithm for the case where Φ and Φ' are of the same polygonal shape. We show that the residuals between two polygonal regions of the same shape, P and P', can be broken into several parts. Each part is attached to one edge and called the edge residual. The computation of $\delta\mathbf{c}(P', P)$ then reduces to the computation of the histogram in every edge residual, which can be pre-computed by a similar incremental scheme as that in (4).

3.1. Edge Residuals

Given a polygonal region P represented by a vertex list $v_1 v_2 \ldots v_N$, the ith edge $v_i v_{i+1}$ ($v_{N+1} = v_1$) is *left-side/right-side* if its left/right side is P's exterior, or *horizontal* if it has a zero slope. Let $v_i = (x_i, y_i)$, and $v_1 v_2 \ldots v_N$ be clockwise ordered, $v_i v_{i+1}$ is left-side if and only if $y_i > y_{i+1}$, and is right-side if and only if $y_i < y_{i+1}$.

As shown in Fig. 1b, we move P horizontally a little rightward with a given stride and obtain a new polygonal region $P' = u_1 u_2 \ldots u_N$. For each non-horizontal edge $v_i v_{i+1}$, an edge residual $\Delta(v_i, v_{i+1})$ is formed by $v_i v_{i+1} u_{i+1} u_i$. The edge residuals attached to right-side edges are parts of the positive residual $P' \backslash P$, and those attached to left-side edges are parts of the negative residual $P \backslash P'$. These edge residuals constitute the overall residual between P and P'. Thus, the over residual histogram

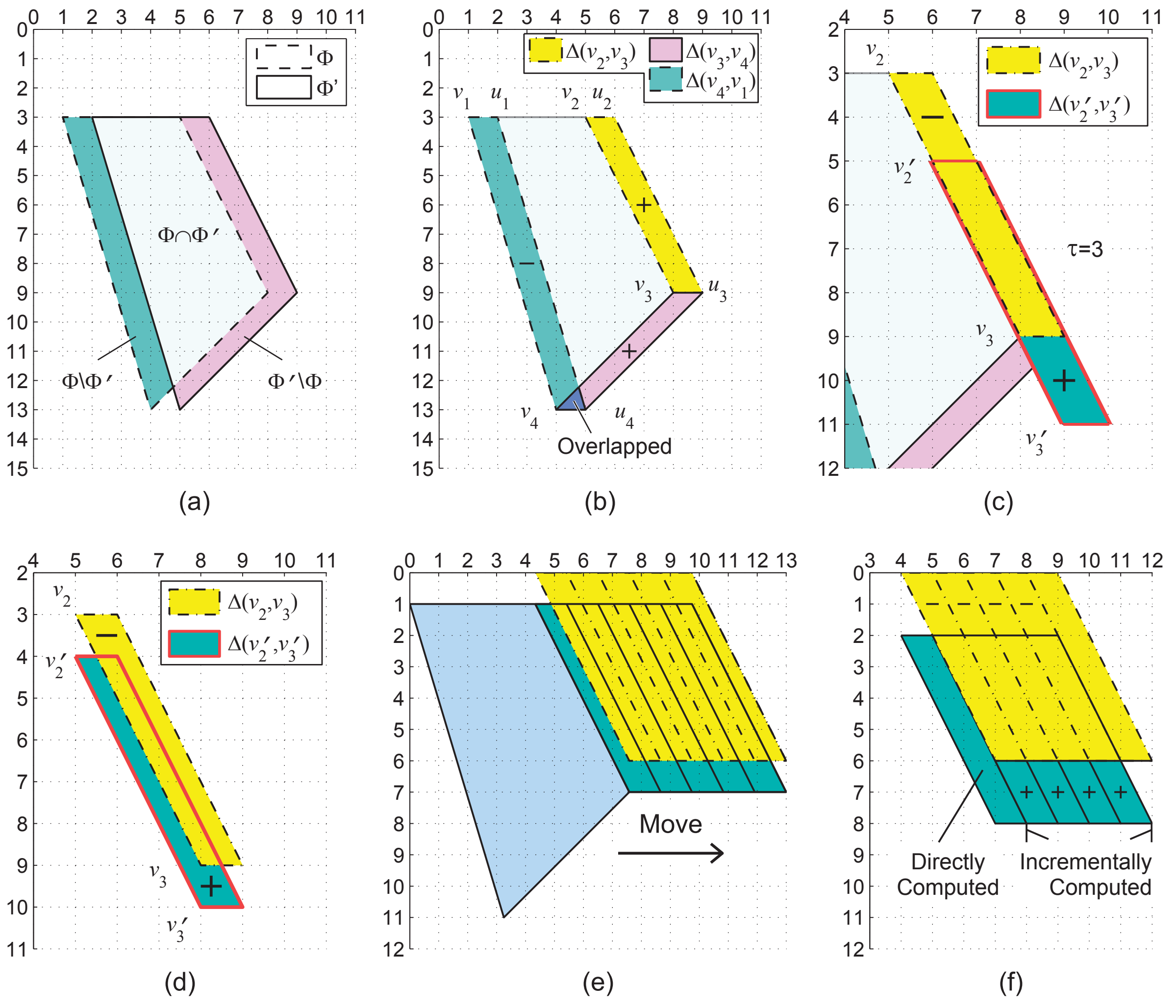


Figure 1: The diagram of incremental histogram computation in polygonal regions. (a) Positive/negative residual. (b) Edge residuals generated by sliding the polygon. (c) The good edge residual pair for incrementally computing the histograms in edge residuals. (d) The bad edge residual pair with small overlaps and irregular non-overlapped regions between them. (e) The histograms in edge residuals in the first a few rows are directly computed. (f) The histograms in most edge residuals are incrementally computed.

$\delta\mathbf{c}(P', P)$ can be computed by

$$\delta\mathbf{c}(P', P) = \sum_{i=1}^{N}(-1)^{\omega_i}\mathbf{c}(\Delta(v_i, v_{i+1})), \tag{6}$$

where $\omega_i = 1$ if $v_i v_{i+1}$ is a left-side edge, otherwise $\omega_i = 0$. Note that the overlap between edge residuals always happens in a left-side edge and a right-side one, so it is neutralized in the computation of $\delta\mathbf{c}(P', P)$.

The decomposition scheme used in (6) has a loosely analogical relationship with that of image integration in [18], where the integration in a polygonal region is decomposed into the summation of those in several triangular regions attached with edges. Nevertheless, while the decomposition scheme in [18] is tailored to polygons, ours is for the residuals between two polygonal regions of the same shape.

3.2. Pre-computation of the Histograms in Edge Residuals

The densely overlapped polygonal regions are generated by sliding a polygon from the origin with a fixed stride to any valid positions on the index bitmap. According to (6), we need to compute the histograms in edge residuals to obtain the overall residual histograms between adjacent overlapped polygonal regions. During the sliding of a polygonal region, each edge generates a sequence of edge residuals, as shown in Fig. 1e. As there are overlaps between them, the histograms in these regions can be pre-computed by a similar incremental manner to that in (4). Previously, Wei and Tao proposed a technique in [17] for histogram computation in rectangular regions with the same purpose as ours. However, our case is more obscure than theirs. For one edge residual, we first discuss how to find a good counterpart to apply the incremental scheme. Then, we give the overall pre-computation procedure. We assume each vertex of a polygon P has an integer coordinate and P slides along the row or column pixel by pixel. The stride will be generalized to arbitrary integer values later.

For an edge $v_i v_{i+1}$, where $v_i = (x_i, y_i)$, let its width be $w_i = |x_{i+1} - x_i|$, its height be $h_i = |y_{i+1} - y_i|$, and $\tau_i = \gcd(w_i, h_i)$, where $\gcd(\cdot, \cdot)$ is the greatest common divisor. As pointed out in [18], v_iv_{i+1} passes $(\tau_i - 1)$ integer points besides its two ends. These integer points equally divide v_iv_{i+1} into τ_i segments, each of which has width $\hat{w}_i = w_i/\tau_i$ and height $\hat{h}_i = h_i/\tau_i$. These segments are the smallest ones ending at integer points, and are also the largest ones internally passing no integer point, so we take them as the *unit segments* of v_iv_{i+1}. For instance, in Fig. 1c, v_2v_3 is equally divided by $(6,5)$ and $(7,7)$ into 3 unit segments with width 1 and height 2.

As shown in Fig. 1c, when v_iv_{i+1} is moved downward to $v'_iv'_{i+1}$ along with the moving of P, there are two edge residuals, $\Delta(v_i, v_{i+1})$ and $\Delta(v'_i, v'_{i+1})$. Provided $\tau_i > 1$, the two edge residuals have the good overlap layout when the coordinate of v'_i is

$$v'_i = (x_i + \alpha\hat{w}_i, y_i + \hat{h}_i), \tag{7}$$

where,

$$\alpha = \text{sign}((y_{i+1} - y_i)\cdot(x_{i+1} - x_i)) \in \{-1, 0, 1\}.$$

In the good layout, the overlapped region and non-overlapped regions between $\Delta(v'_i, v'_{i+1})$ and $\Delta(v_i, v_{i+1})$ all have regular shapes, say parallelograms. Further, because $v'_iv'_{i+1}$ is obtained by sliding v_iv_{i+1} with one of its unit segments, $\Delta(v'_i, v'_{i+1})$ is the edge residual closest to $\Delta(v_i, v_{i+1})$ that has regular overlap layout with $\Delta(v_i, v_{i+1})$. In contrast, if (7) does not hold, the overlap layout is not good, where the non-overlapped regions can be irregular and hard to deal with, as shown in Fig 1d.

By finding the good counterpart for one edge residual, we can use the incremental scheme to improve the efficiency of the histogram pre-computation in all the edge residuals. The overall procedure is summarized as follows. For each edge v_iv_{i+1}, we first directly compute the histograms in all its residuals in the first $\hat{h}_i$ rows (see Fig. 1e). Then, if $\alpha > 0$, the histograms in its residuals in the first $\hat{w}_i$ columns are computed directly (see Fig. 1f). Otherwise, the histograms in its residuals in the last $\hat{w}_i$ columns are directly computed. As the spatial dependency of the incremental computation has an offset of $(\alpha\hat{w}_i, \hat{h}_i)$, the obtained histograms are sufficient initialization for computing the other histograms in v_iv_{i+1}'s residuals. Thus, for each of the rest edge residuals, we find its good counterpart whose histogram has been obtained and compute the histogram incrementally and efficiently by (4) (see Fig. 1f).

The incremental pre-computation of histograms in v_iv_{i+1}'s edge residuals is useful only when $\tau_i > 2$. If $\tau_i = 1$, $\Delta(v_i, v_{i+1})$ and $\Delta(v'_i, v'_{i+1})$ do not have any overlap when (7) holds. If $\tau_i = 2$, the non-overlapped regions between $\Delta(v_i, v_{i+1})$ and $\Delta(v'_i, v'_{i+1})$ has the same size as that of $\Delta(v'_i, v'_{i+1})$. In either case, the time complexity of the incremental pre-computation is not smaller than that of the direct computation. Therefore, we do not pre-compute the histograms in v_iv_{i+1}'s edge residuals by incremental scheme if $\tau_i \le 2$.

Let ξ_x and ξ_y be the horizontal and vertical sliding strides, respectively. With only minor modifications, the proposed algorithm can be adapted to the situation where ξ_x and ξ_y can be larger than 1. Let

$$\kappa_i = \text{lcm}\left(\frac{\xi_x}{\text{gcd}(\hat{w}_i, \xi_x)}, \frac{\xi_y}{\text{gcd}(\hat{h}_i, \xi_y)}\right), \tag{8}$$

where $\text{lcm}(\cdot,\cdot)$ is the least common multiple. It is verified that, when $v'_i - v_i = (\alpha\kappa_i\hat{w}_i, \kappa_i\hat{h}_i)$, $\Delta(v_i, v_{i+1})$ and $\Delta(v'_i, v'_{i+1})$ are in the good layout. In this case, we do not need to pre-compute v_iv_{i+1}'s residuals at every integer position on the index bitmap, as they are arrayed with the horizontal interval ξ_x and vertical interval ξ_y. Note that, the incremental scheme for pre-computing the histograms in v_iv_{i+1}'s edge residuals is only useful when $\tau_i > 2\kappa_i$.

4. Incremental Function Evaluation on Histograms

For many applications such as object detection [1], an objective function using histograms as input needs to be evaluated. As pointed out in [17], if the objective function is bin-additive, i.e., $F(\Phi) = \sum_{b=1}^{B} f_b(c_b(\Phi))$, $F(\Phi')$ can be updated from $F(\Phi)$ by adding a term $\delta F(\Phi', \Phi)$, where

$$\delta F(\Phi', \Phi) = \sum_{b=1}^{B} \delta f_b(\Phi', \Phi), \tag{9}$$
$$\delta f_b(\Phi', \Phi) = f_b(c_b(\Phi')) - f_b(c_b(\Phi)).$$

As previously mentioned, Φ and Φ' denote two overlapped regions. Still, only those entries corresponding to the non-zero entries of $\delta\mathbf{c}(\Phi', \Phi)$ are needed to be computed. In more detail, $f_b(c_b(\Phi))$ have been computed for $b = 1, 2, \ldots, B$. If the bth entry of $\delta c(\Phi', \Phi)$ is zero, we directly get $f_b(c_b(\Phi')) = f_b(c_b(\Phi))$ and $\delta f_b(\Phi', \Phi) = 0$. Otherwise, we evaluate $f_b(c_b(\Phi'))$ and compute $\delta f_b(\Phi', \Phi)$. Note that, if we let $f_b(c) = c\epsilon_b$, where $\{\epsilon_b\}_{b=1}^{B}$ is the standard basis of $\mathbb{R}^B$, $F(\Phi)$ becomes $\mathbf{h}(\Phi)$, which indicates that the histogram computation is a special case of the function evaluation.

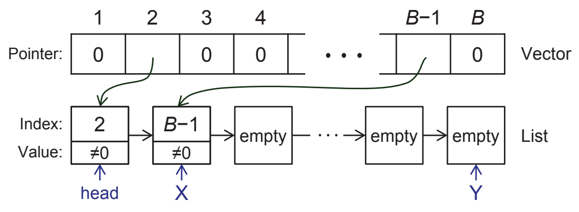
The above extension from the incremental histogram computation to the incremental evaluation of histogram-based function is independent of the shapes of image regions, and hence does not affect the incremental computation scheme for polygonal regions presented in the previous section.

5. Implementation Detail and Algorithm Analysis

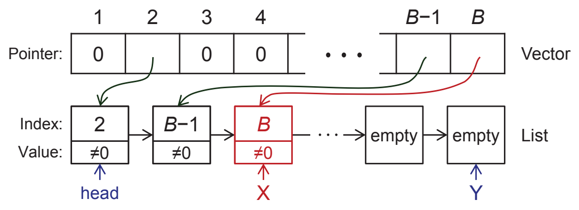
In this section, we first present the data structure for handling the histograms in both edge residuals and overall residuals in a very efficient way, give more detail on the computation procedure, and make a comprehensive analysis on the time complexity of our algorithm. Finally, an additional improvement is also presented.

5.1. Data Structure of Histograms in Residuals

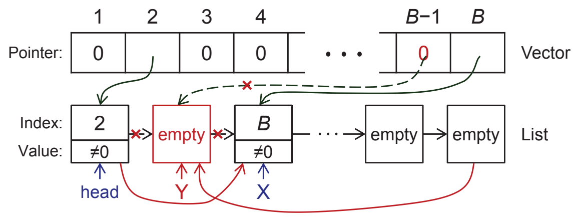
We use a vector-list pair structure to store a histogram in a residual. An illustration of this structure is shown in Fig. 2a. Non-zero entries are stored in a list with a predefined length of no more than the histogram bin number B. Each list node is either left empty or filled with the value and index of one non-zero entry. The non-empty nodes are kept in the front of the list, and the empty ones are left behind. We use a pointer `X` to indicate the end of the non-empty sub-list, and `Y` to that of the whole list. The list head is denoted by `head`. Pointers to the



(a) The vector-list structure.



(b) Node insertion: the *B*th bin is modified to non-zero. The modified part is colored with red.



(c) Node removal: the (*B*−1)th bin is modified to zero. The modified part is colored with red. The red crosses denote the deleted links.

Figure 2: Data structure and histogram operation in our method. X: the end of non-empty nodes. Y: the list end.

list nodes (`NULL` for zeros) are stored in a *B*-entry vector in the same order as that of the histogram bins. When a new non-zero entry emerges, an "insertion" operation is performed, where we assign the entry to the next node of the current `X` and point `X` to it (see Fig. 2b). When a non-zero entry becomes zero, we apply the "removal" operator to the corresponding node, where if the current `X` points to it, we simply move `X` to its previous node; otherwise, we move it to be the next node of the current `Y` and point `Y` to it. Neither "insertion" nor "removal" operator requires memory allocation or deallocation, which is very time-consuming (see Fig. 2c).

Based on above data structure, we can operate the histogram in a very efficient way. Suppose that the time cost of a basic arithmetic operation is t_a. Using the list, the non-zero histogram entries can be quickly traversed in negligible time. Using the vector, a random reference to an entry of the histogram can be done in constant time, which is about t_a and attributes to the arithmetic operation for checking whether the entry is zero-valued. In our implementation, the insertion and removal operators cost the time of about $2t_a$ and $6t_a$, respectively. Thus, the time cost of a random reference to a histogram entry is approximately

$$t_r \approx t_a + s(2t_a + 6t_a)/2 = (1 + 4s)t_a, \qquad (10)$$

where s is the probability that an insertion or removal occurs. The proposed data structure is similar to that in [17]. Nevertheless, since the latter needs linear time in conducting an "insertion" or "removal", ours are more efficient.

5.2. *Computation Detail*

We use the vector-list pair to store edge residuals. For the ith edge of the polygon, $v_i v_{i+1}$, we allocate $W_i \times \hat{h}_i$ vector-list pairs, where $\hat{h}_i$ is the height of its unit segment as previously mentioned in (7), and W_i is the number of $v_i v_{i+1}$'s edge residuals per row. The vector-list pair array is obviously not large enough for storing the histograms in all $v_i v_{i+1}$'s edge residuals at the same time. Indeed, it is also not necessary to pre-compute all the histograms at once.

For the clarity purpose, we only discuss the case in which the polygonal region slides with the stride of 1. We slides the polygonal region from the left to the right along a row, and repeat the horizontal sliding from the top row to the bottom. When the polygonal region slides horizontally, only the histograms in one row of the edge residuals is needed. Hence, we only pre-compute histograms in the edge residuals along each row when the polygonal regions slides there.

However, rather than discard these histograms immediately after the horizontal sliding of the polygon along a row, we should keep them until the histograms depending on them are computed. For $v_i v_{i+i}$, let us write the mapping from the histograms in its residuals to the entries of the vector-list pair array as $\psi_i(x, y)$, where (x, y) is the left-corner location of a residual, and $\psi_i(x, y)$ outputs the column-row index (starts from $(0, 0)$) of the array entry mapped with the histogram in that residual. This mapping is defined as

$$\psi_i(x, y) = \Big(\mathrm{mod}(x + \hat{w}_i \cdot \lfloor y/\hat{h}_i \rfloor, W_i)\, , \, \mathrm{mod}(y, \hat{h}_i) \Big), \qquad (11)$$

where $\lfloor \cdot \rfloor$ is the floor function, and $\mathrm{mod}(\cdot, \cdot)$ computes the remainder left behind upon dividing the first argument by the second one. It is easy to see that $\psi_i(x + \hat{w}_i, y + \hat{h}_i) = \psi_i(x, y)$, which indicates that the two histograms in the residuals of a good layout for incremental pre-computation shares the same vector-list pair. Hence, without allocating a new vector-list pair, the histogram in an edge residual can be directly updated from its counterpart located at some row with a smaller y-coordinate. In additional, as long as $\psi_i(x', y') \neq \psi_i(x, y)$ for $|y' - y| < \hat{h}_i$, the $W_i \times \hat{h}_i$ vector-list array is sufficient large for handling the histograms in all $v_i v_{i+i}$'s edge residuals.

5.3. *Time Complexity*

Based on the vector-list pair structure and the above computational procedure, we now discuss the computational complexity in the three key parts of our method: 1) the histogram pre-computation in edge residuals; 2) the overall residual histogram computation; 3) the function evaluation. Still, we only discuss the case where the sliding stride is 1.

As to the histogram pre-computation in edge residuals, the histogram in an residual of the ith edge, e.g. $\Delta(v'_i, v'_{i+1})$, can be updated from that in its good counterpart, e.g. $\Delta(v_i, v_{i+1})$, by $2r_i$ references to the histogram entries and $2r_i$ additions, where r_i is the number of pixels in one of the two non-overlapped small regions. By saying the pixels *in* a region, we actually means the pixels intersected with the region, since a pixel at the region's border may only have a portion inside the region.

The computational cost in one sliding step is $2r(t_r + t_a)$, where $r = \sum_i^N r_i$, and N is the number of the polygonal edges.

Suppose n pixels in a region show kn different indices in the histogram on average. Let p be the total number of pixels in all edge residuals. For the computation of the overall residual histogram in one sliding step, only kp references into the overall residual histogram and additions are needed, so its time cost is $kp(t_r + t_a)$. In total, the time cost of the two histogram computation stages in one sliding step is $2r(t_r + t_a) + kp(t_r + t_a) = 2(2r + kp)(1 + 2s)t_a$.

In function evaluation, $\mathbf{c}(\Phi')$ is first updated from $\mathbf{c}(\Phi)$ by kp additions, where Φ' and Φ are two overlapped regions as they are used in (4). Then, kp evaluations on f_b and $2kp$ additions are needed for computing $F(\Phi')$. Let t_f be the time cost for evaluating f_b. The time cost here in one sliding step is $kp(t_f + 3t_a)$.

5.4. Additional Improvement

In addition, we propose an improvement on our algorithm. When more than one edges of the polygon have the same height and width, we let them share the histograms in edge residuals. Thus, the cost of histogram pre-computation in edge residuals is cut down linearly with respect to the number of the edges that are identical in both slope and length. As this improvement has no strong connection with the main algorithm, we present it after the time complexity discussion to avoid unnecessary difficulties in analyzing the algorithm.

6. Evaluation

We compare our method with three other approaches. The first one computes histograms by *direct*ly traversing the pixels in polygonal regions. The second one is the generalization of Pham *et al.*'s triangular *integral* image [18] for histogram computation. The third one computes histograms incrementally, but the histograms in edge residuals are directly computed without pre-computation. We denote it as *INC-Basic* and our method as *INC-Edge*.

For the clarity purpose, we still only focus on the pixel-by-pixel sliding case. We first compare the theoretical time complexity among the four competing methods, and prove that the proposed method shows the lowest time complexity. After that, we investigate the practical performance of different methods on natural images. The experimental results coincide with the theoretical analysis, and the proposed method is demonstrated be the most efficient.

6.1. Algorithm Complexity

Table 1 summarized the time complexity of the four algorithms in one sliding step. The histogram computation and function evaluation are separately investigated in analytical forms. In the following, we explain and analyze the time complexity of each algorithm.

Direct method: Let q be the number of pixels in the given polygonal region. The direct method performs q additions to compute the histogram, which cost the time of qt_a. It then estimates f_b for each of the B bins, and sums up the values with B additions to get the evaluation of F. For each bin, the time cost is $(t_f + t_a)$.

Table 1: Algorithm complexity $k, s \ll 1$, $r \ll p \ll q$

Method	Time cost	
	Histogram computation	Function evaluation
Direct	$q \cdot t_a$	$B \cdot (t_f + t_a)$
Integral	$(r + N)B \cdot 2t_a$	$B \cdot (t_f + t_a)$
INC-Basic	$p \cdot (1 + 2s) \cdot 2t_a$	$kp(t_f + 3t_a)$
INC-Edge	$(2r + kp) \cdot (1 + 2s) \cdot 2t_a$	$kp(t_f + 3t_a)$

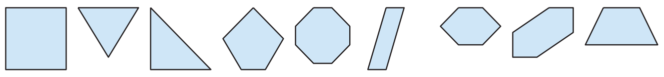


Figure 3: Polygons used in our experiments.

Integral method: In this method, every bin requires a group of triangular integral images. For one bin, the integral image construction at a single position needs about $(r_i + 2\hat{w}_i + 2)$ additions (approximately taken as $2r_i$ for better comparison) for the ith edge of the polygon, where r_i is the same as that used for analyzing INC-Edge. For all the N edges, the time cost is $2rt_a$. With the pre-computed integral images, computing one histogram bin requires $(2N-1)$ (roughly taken as $2N$) additions (refers to [18]). Thus, the time cost for computing histogram is $2(r+N)Bt_a$ in total. The function evaluation here is the same as that of the direct method. Since the time cost is strictly linear to B, the integral method is inefficient for the histograms with large bin numbers.

INC-Basic: Recall the symbols we previously defined for analyzing the proposed algorithm. p is the pixels in all the edge residuals, and k indicates how probable nearby pixels show different index values. In each sliding step of INC-Basic, we need p references into the overall residual histogram and p additions to compute the histogram in the new region. The time cost is $p(t_r+t_a) = 2p(1+2s)t_a$, where s indicates how often a histogram entry switches between zero and non-zero. The function evaluation is the same as that in INC-Edge.

The time complexity of INC-Basic in histogram computation is lower than that of the direct integral method. When the polygon is not too small, q/p is large. As s is a small number, usually in $(0.03, 0.3)$ for natural images [17], $q/p > 2(1 + 2s)$ holds in most of the time, so INC-Basic is more efficient than the direct method in histogram computation.

The comparison on histogram computation between INC-Basic and the integral method is somewhat complex. When the polygon is large, q become large. Thus, it is possible that $p(1+2s) > (r+N)B$, which indicates INC-Basic may not as efficient as the integral method for large polygons. However, p is practically too small so that the opposite of the above inequality nearly always holds in practice. In Section 6.2, we will demonstrate that even when the polygon is much larger than the nor-

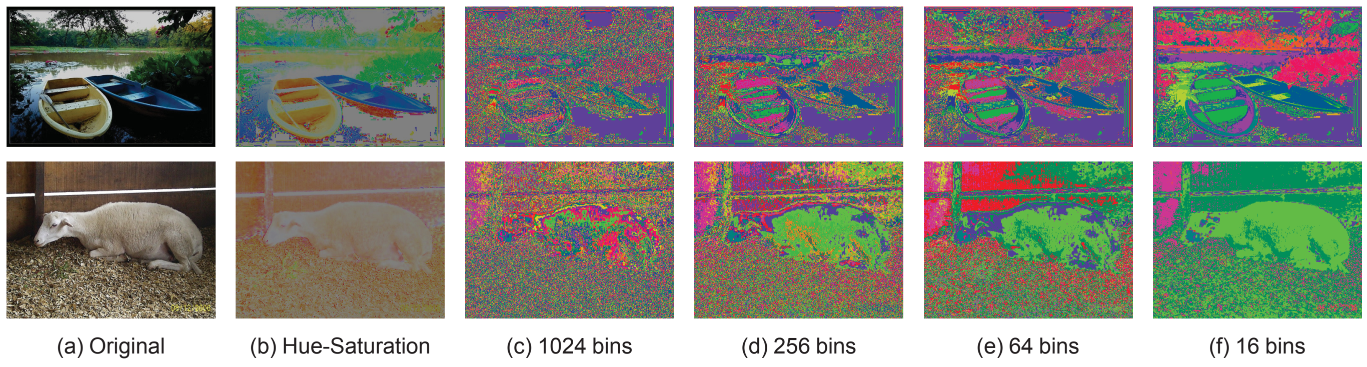


Figure 4: Illustrations of images in our experiments. a: original RGB images; b: the hue-saturation images with constant intensity; c-f: uniformly quantized hue-saturation images with various quantization levels, where each different value maps to a unique color.

mally used size, INC-Basic still significantly outperforms the integral method in efficiency.

In the function evaluation, k is usually very small because of the local smoothness in natural images. According to [17], k is normally in the same range as s, say approximately $(0.03, 0.3)$. Consequently, $kp \ll B$ generally holds. It indicates that INC-Basic is more efficient in function evaluation than the direct and integral method.

INC-Edge: INC-Edge is the same as INC-Basic in function evaluation, and more efficient than INC-Basic in histogram computation.

Let τ_0 be the greatest number satisfying $\tau_i \geq \tau_0$ for $i = 1, 2, \ldots, N$. We have $(2r + kp) \leq (k + 2/\tau_0)p$. Because $k \ll 1$ as previously mentioned, if $\tau_0 > 2$, it holds $(k + 2/\tau_0) < 1$, which indicates that INC-Edge is more efficient than INC-Basic. Besides, even if $\tau_0 \leq 2$, the efficiency of INC-Edge is at least the same as that of INC-Basic, as long as we always switch the way of pre-computing the histograms in the edge residuals of the ith edge from incremental scheme to the direct method if $\tau_i \leq 2$ (refer to Section 3.2). In addition, as polygons with large τ_0 can always be designed, the efficiency of INC-Edge can be improved further.

6.2. *Quantitative Evaluation*

In our experiments, we use the nine predefined polygons shown in Fig. 3, and set the sliding stride to 1.

To practically evaluate the efficiency of our methods, we randomly chose 100 natural images from PASCAL VOC2010 dataset [20] for testing. The images are converted into the HSL color space. Their hue and saturation are uniformly quantized into bins for histogram computation. As it is shown in Fig. 4, the natural images that we use show local smoothness (Fig. 4a), which is more significant when only the hue and saturation are kept (Fig. 4b). In the index bitmaps quantized from the hue-saturation images, identical values occur frequently among nearby pixels (Fig. 4c-f). This property agrees with the assumption of our method. As it is pointed out in [17], many other types of image responses, such as quantized gradient orientation [1], also have such a property.

For the evaluation of objective functions, we compared three functions, dot product, entropy and SVM χ^2 kernel, whose computational cost grows one by one. These functions are useful in solving many computer vision problems. The dot product simulates linear classifiers, such as linear SVM and AdaBoost. Entropy is useful in tasks such as saliency detection, and may also simulate some regression kernels. The χ^2 kernel is a popular kernel for non-linear SVM. We omit the analytical forms of the three functions here, as [17] have addressed them in detail.

We implemented the competing methods, say the direct method, integral method, INC-Basic, and INC-Edge, in standard C++ with the single-thread paradigm. For fair comparison, our implementation of the four methods shares the same auxiliary modules, such as those for manipulating polygons and evaluating objective functions. We compiled the implementation by GNU C++ Compiler with Level 3 optimization (-O3), a commonly used setting for building UNIX applications, and carried out the experiments on a common PC with a 3.2GHz Intel Core i3 CPU and 4G 1333MHz RAM.

We measure the algorithm efficiency in *Time Consumption Per Region* (**TCPR**), which is the average time spent on computing the histogram in each image region on all the test images. Results on the nine different polygons are averaged.

In Fig. 5, we compare the efficiency between our method and the other methods when the polygon size changes, where the size of a polygon is defined as the maximum of the polygon's width and height. The TCPR of the direct method grows very fast with the increase of the polygon size. That of the integral method appears stable, but always stays at a high cost level. Even though the TCPR of INC-Basic and INC-Edge grows, it is still much lower than that of the integral method when the polygons become very large. As we expected, INC-Edge achieves the best efficiency in both the histogram computation and function evaluation. Its TCPR ranges from 10^{-6} to 10^{-5} second for function evaluation, which indicates that densely evaluating histogram-based function on a 480×360 image can be processed in only 1 second even in large image regions.

Fig. 6 shows the TCPRs of the competing methods against the bin number of histograms. Our algorithm is more efficient than the direct method by an order of magnitude. Compared with the integral method, the TCPR increase of our method is much slower against the growth of the bin number. The gap between INC-Edge and INC-Basic decreases as the bin number

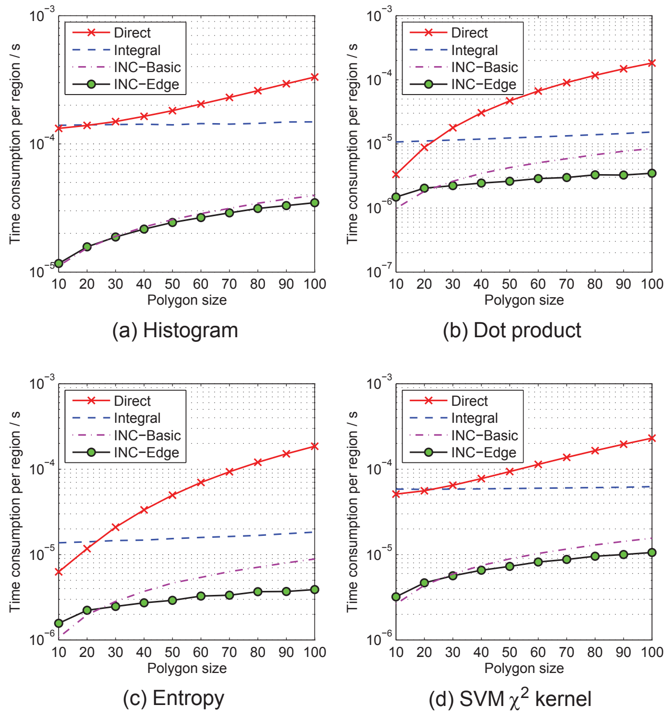


Figure 5: Time consumption against polygon size (measured by TCPR in seconds). The bin number is 256.

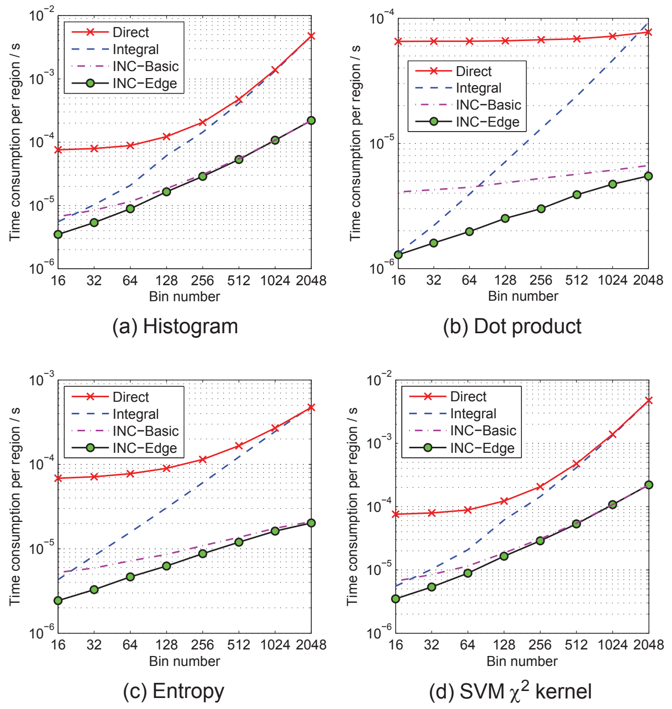


Figure 6: Time consumption against bin number (measured by TCPR in seconds). The polygon size is 60.

grows. A probable interpretation is that the function evaluation contributes more to the overall time complexity when the bin number becomes larger, so INC-Edge's superiority over INC-Basic in histogram computation turns out to be less significant. Nevertheless, in the normally used range of the histogram bin number, INC-Edge outperforms INC-Basic.

7. Conclusion & Future Work

Research on efficient computation of histograms in non-rectangular regions is important, but have not been got enough attention. We have proposed an incremental computation method to efficiently compute the histograms in polygonal regions. Compared with state-of-the-art methods, our method achieves much higher efficiency in both the histogram computation and function evaluation. We believe that this algorithm have solved one of the biggest obstacles in using histograms computed in polygonal regions. With the proposed algorithm, research topics such as polygonal local histogram-based features and non-rectangular object detection/recognition can be studied in the further. It is also possible to generalize the proposed algorithm for histogram-based feature extraction in non-cubic 3-D volumes, which may benefit histogram-based approaches for video classification [21] and 3-D face recognition [22].

Acknowledgment

This work was partly supported by National 973 Program (2013CB329504), National Natural Science Foundation of China (No. 61070067, No. 61103107), and Research Fund for the Doctoral Program of Higher Education of China (No. 20110101120154).

[1] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005, Vol. 1, 2005, pp. 886 –893 vol. 1. doi:10.1109/CVPR.2005.177.
[2] D. Comaniciu, V. Ramesh, P. Meer, Real-time tracking of non-rigid objects using mean shift, in: IEEE Conference Proceedings on Computer Vision and Pattern Recognition, 2000., Vol. 2, 2000, pp. 142 –149 vol.2. doi:10.1109/CVPR.2000.854761.
[3] T. Ojala, M. Pietikainen, T. Maenpaa, Multiresolution gray-scale and rotation invariant texture classification with local binary patterns, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (7) (2002) 971 –987. doi:10.1109/TPAMI.2002.1017623.
[4] D. G. Lowe, Distinctive image features from scale-invariant keypoints, International Journal of Computer Vision 60 (2004) 91–110. doi:10.1023/B:VISI.0000029664.99615.94.
[5] T. Leung, J. Malik, Representing and recognizing the visual appearance of materials using three-dimensional textons, International Journal of Computer Vision 43 (2001) 29–44. doi:10.1023/A:1011126920638.
[6] P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan, Object detection with discriminatively trained part-based models, IEEE Transactions on Pattern Analysis and Machine Intelligence 32 (9) (2010) 1627 –1645. doi:10.1109/TPAMI.2009.167.
[7] T. Ahonen, A. Hadid, M. Pietikainen, Face description with local binary patterns: Application to face recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (12) (2006) 2037 –2041. doi:10.1109/TPAMI.2006.244.
[8] L. Fei-Fei, P. Perona, A Bayesian hierarchical model for learning natural scene categories, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005., Vol. 2, 2005, pp. 524 – 531 vol. 2. doi:10.1109/CVPR.2005.16.

[9] Y. Wei, J. Sun, X. Tang, H.-Y. Shum, Interactive offline tracking for color objects, in: IEEE 11th International Conference on Computer Vision, 2007. ICCV 2007., 2007, pp. 1 –8. doi:10.1109/ICCV.2007.4408949.

[10] C. Liu, J. Yuen, A. Torralba, SIFT flow: Dense correspondence across scenes and its applications, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (5) (2011) 978 –994. doi:10.1109/TPAMI.2010.147.

[11] G. Qiu, Indexing chromatic and achromatic patterns for content-based colour image retrieval, Pattern Recognition 35 (8) (2002) 1675 – 1686, colour Imaging. doi:10.1016/S0031-3203(01)00162-5.

[12] G. Pan, L. Sun, Z. Wu, Y. Wang, Monocular camera-based face liveness detection by combining eyeblink and scene context, Telecommunication Systems 47 (2011) 215–225. doi:10.1007/s11235-010-9313-3.
URL `http://dx.doi.org/10.1007/s11235-010-9313-3`

[13] G. Pan, Y. Zhang, Z. Wu, Accelerometer-based gait recognition via voting by signature points, Electronics Letters 45 (22) (2009) 1116 –1118. doi:10.1049/el.2009.2301.

[14] F. Porikli, Integral histogram: a fast way to extract histograms in Cartesian spaces, in: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2005. CVPR 2005, Vol. 1, 2005, pp. 829 – 836 vol. 1. doi:10.1109/CVPR.2005.188.

[15] F. C. Crow, Summed-area tables for texture mapping, SIGGRAPH Computer Graphics 18 (1984) 207–212. doi:10.1145/800031.808600.

[16] M. Sizintsev, K. Derpanis, A. Hogue, Histogram-based search: A comparative study, in: IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008, 2008, pp. 1 –8. doi:10.1109/CVPR.2008.4587654.

[17] Y. Wei, L. Tao, Efficient histogram-based sliding window, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010, 2010, pp. 3003 –3010. doi:10.1109/CVPR.2010.5540049.

[18] M.-T. Pham, Y. Gao, V. Hoang, T.-J. Cham, Fast polygonal integration and its application in extending haar-like features to improve object detection, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2010, 2010, pp. 942 –949. doi:10.1109/CVPR.2010.5540117.

[19] T. Huang, G. Yang, G. Tang, A fast two-dimensional median filtering algorithm, IEEE Transactions on Acoustics, Speech and Signal Processing 27 (1) (1979) 13 – 18. doi:10.1109/TASSP.1979.1163188.

[20] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, A. Zisserman, The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results, http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[21] I. Laptev, M. Marszalek, C. Schmid, B. Rozenfeld, Learning realistic human actions from movies, in: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on, 2008, pp. 1 –8. doi:10.1109/CVPR.2008.4587756.

[22] Y. Wang, X. Tang, J. Liu, G. Pan, R. Xiao, 3D face recognition by local shape difference boosting, in: Computer Vision – ECCV 2008, Vol. 5302 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 603–616. doi:10.1007/978-3-540-88682-2_46.