Min-Space Integral Histogram

Anonymous ECCV submission

Paper ID 245

Abstract. In this paper, we present a new approach for quickly computing the histograms of a set of unrotating rectangular regions. Although it is related to the well-known Integral Histogram (IH), our approach significantly outperforms it, both in terms of memory requirements and of response times. By preprocessing the region of interest (ROI) computing and storing a temporary histogram for each of its pixels, IH is effective only when a large amount of histograms located in a small ROI need be computed by the user. Unlike IH, our approach, called *Min-Space Integral Histogram*, only computes and stores those temporary histograms that are strictly necessary (less than 4 times the number of regions). Comparative tests highlight its efficiency, which can be up to 75 times faster than IH. In particular, we show that our approach is much less sensitive than IH to histogram quantization and to the size of the ROI.

1 Introduction

The complex nature of images implies that a large amount of data needs to be stored in histograms (colors, edges, *etc.*), hence making their computations time consuming. In many applications, large sets of histograms need be computed frequently, hence making it a computational bottleneck. This explains why fast histogram computation has received some attention in the literature [1–4].

An image yields a distribution over a color space by mapping each of its pixels into its color. The histogram H of an $N \times M$ image $\mathcal{I}$ is defined by $H(k) = \sum_{x=1}^{N} \sum_{y=1}^{M} \{\mathcal{I}(x,y) = k\}$, where $\mathcal{I}(x,y)$ is a pixel, $k = 0, \ldots, K-1$ is its value (in this paper, this is a color value, but gray intensities, gradient orientations, etc., could also be considered). Binning the probability distribution induced by H is a way to summarize it, and the applied quantization (bin size) controls the rate of summarization. In such a case, histogram H is divided into B bins $b = 0, \ldots, B-1$, and is defined by:

$$H(b) = \sum_{x=1}^{N} \sum_{y=1}^{M} \left\{ \mathcal{I}(x,y) \in \left[b\frac{K}{B}, (b+1)\frac{K}{B} \right[\right\}.$$

The classical approach to compute an histogram of a $w \times h$ region R consists of browsing all its pixels, hence yielding a time complexity of $O(wh)$.

In this paper, we propose a novel approach to speed-up multiple histogram computation by reducing computational redundancies. We will show that this new method outperforms the well-known Integral Histogram (IH) both in terms of memory requirements and of response times. The paper is organized as follows. Section 2 first presents a short overview of the main approaches that have been proposed in the literature to speed-up histogram computation. In particular, it recalls IH's approach. Section 3 then describes our new approach: we first present the overview of the method and, then, we detail it, including its time and space complexity. Section 4 gives some comparative results for the computation of a set of histograms, both in terms of response times and memory requirements. Three approaches are compared; the classical one, IH and our method. Finally, concluding remarks are given in Section 5.

2 Fast Histograms Computations

The classical approach would certainly be sufficient for practical applications did the latter need computing only very few histograms. Unfortunately, in practice, applications often have to repeatedly compute large sets of histograms and, in those cases, a more efficient approach is compulsory to get admissible response times. To achieve this, it can be observed that the rectangular regions where histograms are computed often overlap, thus inducing some redundancies. Exploiting the latter is the key idea underlying fast histograms computation.

One of the first works in this direction was proposed in [1], in the context of image filtering (median filter). Considering the histogram H_R of a region R, that of another region Q is computed by removing from H_R the histogram of the pixels that belong to R but not to Q and adding that of the pixels that belong to Q but not to R. This approach can be very efficient when the two regions considered have a large intersection. Similar in spirit, the method proposed in [2] breaks up region R into the union of its columns in the image, and all the column histograms are kept up to date in constant time using a two-step approach. In [3], the authors propose the distributive histogram based on a distributive property of disjoint regions combined with a per-column histogram maintenance and a row-based update of these column histograms. This approach can be easily extended to cope with non-rectangular regions and multi-scale processing.

When massive amounts of histograms need be computed, IH [4] proves to be particularly effective and, actually, it is now used in many practical applications, especially in recent tracking algorithms [5]. Consider a set of regions $\{R_1, \ldots, R_n\}$ where histograms need be computed. We will call them "goal histograms" (GH). Each region R_i is identified by a quadruple $\langle x_i, y_i, w_i, h_i \rangle$ where (x_i, y_i) are the coordinates of the bottom right corner of the region in the image and w_i and h_i refer to the width and height of the region respectively. Instead of computing directly the goal histograms, IH preprocesses the image to compute efficiently a set of "temporary histograms" (TH) that prove to be sufficient to compute all the GHs. More formally, let R be the smallest rectangular area containing all the R_i, $i = 1, \ldots, n$, i.e., R is the image's region of interest (ROI). IH's preprocess consists of computing for each pixel p of the ROI the histogram $TH(p)$ of the upper left region of p. To perform this efficiently, it exploits the following formula, for any pair of coordinates (x, y) of the ROI:

$$TH(x, y) = \mathcal{I}(x, y) + TH(x-1, y) + TH(x, y-1) - TH(x-1, y-1). \quad (1)$$

THs are computed from the upper left corner of the ROI to the bottom right one and, thus, each TH is inferred from the previously computed THs using only three arithmetic operations. This guarantees the efficiency of the method. Once this preprocess is completed, for each region R_i, $i = 1, \ldots, n$, the goal histogram H_{R_i} of region R_i is simply computed as:

$$H_{R_i} = TH(x_i, y_i) - TH(x_i - w_i, y_i) - TH(x_i, y_i - h_i) + TH(x_i - w_i, y_i - h_i). \quad (2)$$

Again, the three operations involved in Eq. (2) make IH particularly effective. However, IH has two major drawbacks. First, if the ROI is large, IH consumes a large amount of memory to store one TH per ROI's pixel. Actually, if the ROI is an $N \times M$ region and if B is the total number of bins per histogram[1], then, the memory used by the THs is of size $N \times M \times B$. Second, when the number of GHs is relatively small, the classical approach that directly computes all the GHs significantly outperforms IH as the latter needs to compute many THs. In the next section, we will propose a new approach addressing both problems.

Note that all the methods presented above try to speed-up histogram computations by reducing the redundancies between the computations of the GHs in the current image (see [3] for a comparative study). But other types of redundancies can also be exploited, as in [6] where, relying on the spatial differences arising between consecutive frames, the Temporal Histogram proves to be quite effective. In the latter, to compute an histogram in a given frame, the corresponding histogram in the preceding frame is simply updated taking into account only the differences between the preceding frame and the current one. In our paper, we focus on the first type of redundancy: that between the regions of the current image and we provide in the next section a novel algorithm that significantly outperforms both IH and the classical algorithm in most cases.

3 Min-Space Integral Histogram

The Min-Space Integral Histogram (MSIH) is quite similar in spirit to IH. Actually, it computes some THs that correspond to histograms of the upper left region of some pixels. However, unlike IH, it does not store one histogram per pixel in the ROI but rather one per TH that is involved in Eq. (2). In other words, MSIH only requires the THs of the pixels corresponding to the corners of regions R_i[2]. In addition to these points, it also uses one additional temporary histogram during its whole preprocess. Hence it never computes more than $(4n+1)$ THs, thus significantly reducing the memory consumption compared to IH. In addition, this reduction also induces a significant speed-up.

---

[1] When the image contains several channels, B refers to the product of the number of bins per channel, e.g., $B = 512$ for 3 channels of 8 bins each.

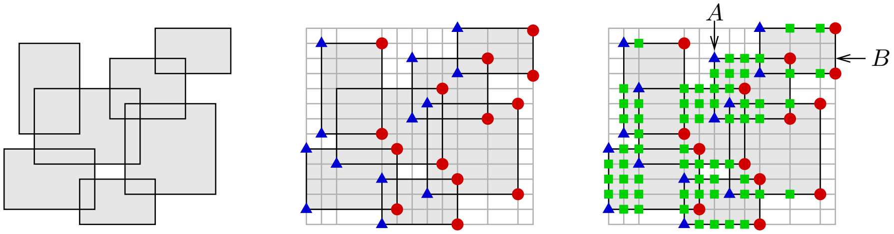
[2] To be precise, like IH, for each region $R_i = \langle x_i, y_i, w_i, h_i \rangle$, MSIH uses the THs corresponding to the points of coordinates (x_i, y_i), $(x_i - w_i, y_i)$, $(x_i, y_i - h_i)$ and $(x_i - w_i, y_i - h_i)$ in the image, which do not correspond exactly to the corners of R_i.

3.1 Overview of the Method

The basic idea of MSIH is summarized in Fig. 1. Consider we wish to compute the histograms of the seven regions of Fig. 1.a and assume that the ROI is a 100×100-pixel area. Then IH has to store 10000 THs although only those corresponding to the points in blue and red on Fig. 1.b, i.e., 28 THs, are actually used in Eq. (2) to compute the histograms of the seven areas. MSIH focuses on these 28 THs. To do so, it first determines the grid where these points are located (the gray lines in Fig. 1.b). Note that this grid contains only 12 rows and 14 columns, hence an overall of 168 cells instead of 10000 for the ROI. Then, MSIH parses this grid from left to right and from top to bottom and computes incrementally the THs in an Integral Histogram-like manner. The basic idea is the following: let $TH(i,j)$ denote the TH at the ith row and jth column of the grid, then:

$$TH(i,j) = \sum_{(x,y)\in G_{ij}} \mathcal{I}(x,y) + TH(i-1,j) + TH(i,j-1) - TH(i-1,j-1), \quad (3)$$

where G_{ij} is the rectangular area of the image defined by the extremal points $(G_X(i-1)+1, G_Y(j-1)+1)$ and $(G_X(i), G_Y(j))$, with $G_X(i)$ and $G_Y(j)$ representing the X and Y-coordinates in the image of the pixel corresponding to the ith row and jth column of the grid respectively. This simple scheme already outperforms IH both in terms of memory consumption (only 168 THs are stored instead of 10000) and in terms of response time (since it involves much fewer arithmetic operations on histograms). But this basic idea can be refined because, computing $TH(i,j)$ for every row and column of the grid is not necessary. Actually, as we will see, computing only those THs that are represented as green squares, blue triangles and red circles on Fig. 1.c is sufficient to compute the histograms of seven regions R_i of interest. This results in a significant improvement as there are only 91 squares, triangles and circles compared to 168 cells on the grid (12 rows by 14 columns). Note that the green points just correspond to points where histogram computations are needed, they do not involve any memory requirement: as mentioned before, at most $(4n+1)$ THs will be kept in memory (here only 29 THs are stored). Finally, we will see that even Eq. (3) itself can be improved so that it involves only 2 arithmetic operations over histograms instead of three, hence further speeding up the method.



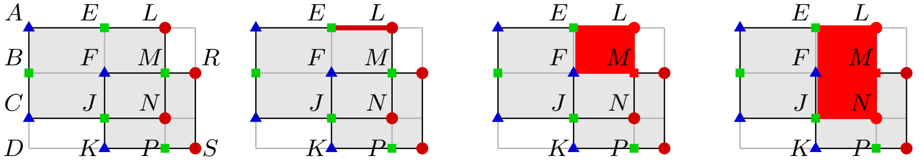
a) 7 regions b) MSIH's grid c) THs and GHs

Fig. 1. The key idea of MSIH.

3.2 Determination of the Temporary Histograms

Eq. (3) is the stepping stone of our TH computation: to the values of the pixels of a rectangular region G_{ij} are added or subtracted 3 previously computed THs: those of the regions above and to the left of G_{ij} and the TH of their intersection. So, the green squares shall be determined in order to enforce that whenever a region G_{ij} is involved in Eq. (3), these three THs do exist. To understand how this can simply be achieved, consider Fig. 2.a. The blue triangles and red circles correspond to the left and right corners of the regions respectively. Consider the computation of S's TH. If THs are computed from left to right and from top to bottom, then those of N, K and R have already been computed. Thus, to avoid parsing pixels of the image more than once, we shall combine the previously computed THs and add to this combination the pixels of the polygon $NMRSKJ$. Unfortunately, this one is not a rectangle, hence ruling out an application of Eq. (3). However, if points M and P are introduced and their THs are computed, then that of S can be computed as $TH(S) = \sum_{(x,y)\in G_S} \mathcal{I}(x,y) + TH(R) + TH(P) - TH(M)$, where G_S denotes rectangle $SPMR$. Note that M and P are located on the same rows as R and S but on the column to their left. Now, consider the computation of $TH(N)$. As mentioned above, this one occurs after the THs of C, F and M have been computed. But, as above, polygon $CFMN$ is not a rectangle. So, to exploit Eq. (3), we shall add a new green square J. Then, $TH(N) = \sum_{(x,y)\in G_N} \mathcal{I}(x,y) + TH(M) + TH(J) - TH(F)$, where G_N denotes rectangle $NJFM$. For the same reason, the computation of $TH(M)$ requires creating new point E. More generally, this suggests that whenever a TH needs be computed at point (i, j), that at point $(i-1, j)$ shall be computed as well. If the latter is not a corner of a region R_i of interest, then it shall be a green square. However, this rule applies only to the points of the grid that belong to some region R_i: those that are outside all the regions need not be computed. For instance, on Fig. 2.a, although the TH of K is needed, that of D is clearly unnecessary. This rule is general and is precisely that applied on Fig. 1.c to determine the 63 green squares. This leads to Algorithm 1 whose correctness is proved in Proposition 1.

Proposition 1. *The grid resulting from Algorithm 1 is such that the THs of all its nonempty points, i.e., those in red, blue or green, can be computed parsing the*



a) green squares b) computing $TH[L]$ c) computing $TH[M]$ d) computing $TH[N]$

Fig. 2. Computing green squares and the THs of a given column.

```
   Input: a set of regions $\{R_1, \ldots, R_n\}$, a $w \times h$ grid $G$
1  initialize $G$: all cells are set to “empty” points (neither green nor blue nor red)
   // create 2 vectors $V_T, V_B$ counting, for each row, the number of
   // top and bottom $R_i$'s corners processed yet on that row
2  $V_T, V_B \leftarrow$ vectors of size $h+1$ filled with 0's
   // fill the grid from right to left
3  for $i = w$ downto 1 do
       // if there is a column on the right, create green the squares:
4      if $i \neq w$ then
5          nb_current_regions $\leftarrow 0$
6          for $j = 1$ to $h$ do
7              nb_current_regions $\leftarrow$ nb_current_regions $+ V_T[j] - V_B[j]$
8              if nb_current_regions $> 0$ and $G[i+1, j] \neq$ empty point then
9                  $G[i, j] \leftarrow$ green square
       // fill $G$ with the regions whose corners are on its $i$th column
10     $\mathcal{R} \leftarrow$ {regions $R_k$ whose right sides are located on the $i$th column of $G$}
11     foreach $R_k = \langle x_k, y_k, w_k, h_k \rangle \in \mathcal{R}$ do
12         convert image coordinates $(y_k, y_k - h_k)$ into grid coordinates $(b_k, t_k)$
13         $G[i, b_k] \leftarrow$ red circle;   $G[i, t_k] \leftarrow$ red circle
14         $V_T[t_k] \leftarrow V_T[t_k] + 1$;   $V_B[b_k + 1] \leftarrow V_B[b_k + 1] + 1$
15     $\mathcal{L} \leftarrow$ {regions $R_k$ whose left sides are located on the $i$th column of $G$}
16     foreach $R_k = \langle x_k, y_k, w_k, h_k \rangle \in \mathcal{L}$ do
17         convert image coordinates $(y_k, y_k - h_k)$ into grid coordinates $(b_k, t_k)$
18         $G[i, b_k] \leftarrow$ blue triangle;   $G[i, t_k] \leftarrow$ blue triangle
19         $V_T[t_k] \leftarrow V_T[t_k] - 1$;   $V_B[b_k + 1] \leftarrow V_B[b_k + 1] - 1$
```

Algorithm 1: Determination of the points where THs are computed.

pixels of the ROI at most once and using Eq. (3) where only the THs of points belonging to regions $R_1, \ldots, R_n$ *are taken into account.*

Proof. We shall first prove by induction that, in lines 5–9, nb_current_regions counts precisely the number of regions R_k whose right and left corners are located in columns $i+1, \ldots, w$ and $1, \ldots, i$ respectively. Of course, at the beginning of the algorithm, this property holds since V_T and V_B are initialized with zeros. Assume that the property holds on the $(i+1)$th rightmost columns of the grid and let us show that it also holds on the ith one. Vectors V_T and V_B are updated on lines 14 and 19: whenever the right side of a new region R_k is encountered (lines 10–14), $V_T[t_k]$ and $V_B[b_k+1]$ are incremented, where t_k and b_k are the grid Y-coordinates of the top and bottom corners of R_k respectively. Thus, within loop 6–9, the increment of $V_T[t_k]$ and $V_B[b_k+1]$ will induce on line 7 that nb_current_regions will be incremented by 1 as well only on rows $t_k, \ldots, b_k$, i.e.,on the rows where region R_k is located. Similarly, when the left side of region R_k is encountered, i.e., on loop 16–19, line 19 will do the inverse process, i.e., it will decrease by 1 nb_current_regions on all the rows $t_k, \ldots, b_k$. Hence, overall, nb_current_regions precisely counts the number of regions whose right sides have already been examined but not their left sides yet.

Thus, green squares are constructed on line 9 whenever i) there exist regions whose right sides have been encountered in the columns already processed, i.e., on the right, but whose left sides have not been encountered yet; and ii) in the right column, there exists either a green, blue or red point. Condition ii) indicates the we shall systematically construct green squares on columns on the left of nonempty points and condition i) removes only those green points that would be created on areas of the grid where no region R_k is located (like point D in Fig. 2.a). Consequently, if a nonempty point, say S has another point R above it (see Fig. 2.a), then, on their left column, line 9 will create two points P and M unless those are outside any region R_k (like point D). Thus, Eq. (3) can be applied and the proposition holds. □

We shall now see how the THs can be computed efficiently on the grid, minimizing the number of arithmetic operations over histograms.

3.3 Efficient Computation of the Temporary Histograms

As suggested by Eq. (3), we shall construct the THs on the grid from left to right and from top to bottom. But applying directly Eq. (3), as would be done to construct the THs of IH, is not efficient because this would involve 3 operations over histograms whereas the same result can be obtained with only 2. To achieve this, let H_{col} be a column vector of histograms of size the number of rows of the grid and let H_{tmp} be an histogram. Assume that, on Fig. 2.a, H_{col} contains the THs of points E, F, J, K, i.e., $H_{col}[1] = TH[E]$ (resp. $H_{col}[2] = TH[F], H_{col}[3] = TH[J], H_{col}[4] = TH[K]$) is the histogram of the union of the subparts of the regions R_k that lie above and to the left of E (resp. F, J, K). Let us now compute the THs of L, M, N, P. First, each time we process a new column in the grid, H_{tmp} is cleared so that it contains no pixel. Now, add to H_{tmp} the pixels of line $]E, L]$ as shown in red on Fig. 2.b. Then, to be a valid TH for L, we should add to H_{tmp} the histogram of the area on the left of E, which corresponds precisely to $H_{col}[1]$. Thus, adding H_{tmp} to $H_{col}[1]$, the latter contains $TH[L]$. Next, add to H_{tmp} the pixels of rectangle $MFEL$ (excluding lines $[E, L]$ and $[E, F]$). Then H_{tmp} contains the histogram of rectangle $MFEL$ excluding only line $[E, F]$ as shown in Fig. 2.c. So, as $H_{col}[2] = TH[F]$ contains the histogram of the area above and to the left of F, after adding H_{tmp} to $H_{col}[2]$, the latter contains $TH[M]$. Similarly, after adding to H_{tmp} the pixels of rectangles $NJFM$ (excluding lines $[F, M]$ and $[F, J]$), H_{tmp} contains the histogram of rectangle $NJEL$ excluding line $[E, J]$ (see Fig. 2.d). Hence, adding H_{tmp} to $H_{col}[3] = TH[J]$, we get $TH[N]$. By processing as shown above, we thus reduce the number of arithmetic operations over histograms. This leads to Algorithm 2 whose correctness is shown in Proposition 2.

Proposition 2. *Applying Eq. (2) with the THs resulting from Algorithm 2 yields correct histograms* H_{R_i}.

Proof. First, note that adding any histogram Δ_1 to both $TH(x_i, y_i)$ and $TH(x_i - w_i, y_i)$ and adding any histogram Δ_2 to $TH(x_i, y_i - h_i)$ and $TH(x_i - w_i, y_i - h_i)$,

```
   **Input**: an image $\mathcal{I}$, a $w \times h$ grid $G$
   **Output**: the set $TH$ of temporary histograms of the blue and red points of $G$
1  $H_{col} \leftarrow$ a vector of size $h$ of empty histograms
2  $H_{tmp} \leftarrow$ an empty histogram;    $TH \leftarrow \emptyset$
3  for $i = 1$ to $w$ do
4      clear $H_{tmp}$
5      for $j = 1$ to $h$ do
6          if $G[i,j]$ is a red circle, a blue triangle or a green square then
7              let $R$ be the rectangle from the last rows and columns where THs
               were computed in the grid (e.g., $NJFM$ for $H_{col}[3]$ in Fig. 2)
8              add to $H_{tmp}$ the pixels of rectangle $R$
9              add $H_{tmp}$ to $H_{col}[j]$
10             if $G[i,j]$ is a blue triangle or a red circle then
11                 save $H_{col}[j]$ into $TH[i,j]$
12                 if there exists no nonempty point on $G[k,j]$ for $k > i$ then
13                     delete $H_{col}[j]$ from memory
14 return the set $TH$ of histograms
```

Algorithm 2: Computation of the Temporary Histograms.

then, in Eq. (2), H_{R_i} is unaffected. So, in this proof, a temporary histogram $TH[i,j]$ is said to be valid if it represents the histogram of the area above and to the left of point $G[i,j]$ minus some histogram Δ and, for all points $G[k,j]$ such that the whole segment $[G[i,j], G[k,j]]$ belongs to $\bigcup_{k=1}^{n} R_k$, $TH[k,j]$ represents the histogram of the area above and to the left of point $G[k,j]$ minus Δ. For instance, in Fig. 1.c, all the THs on segment $[A,B]$ should subtract the same Δ. This is sufficient to ensure that the same Δ will be removed from both $TH(x_i, y_i)$ and $TH(x_i - w_i, y_i)$ (resp. $TH(x_i, y_i - h_i)$ and $TH(x_i - w_i, y_i - h_i)$) in Eq. (2).

The rest of the proof is by induction. For the leftmost column of the grid, after executing lines 4–13, H_{col} clearly contains the THs of every nonempty grid point of the column since, at each iteration, $H_{col}[j] = H_{tmp}$ and, by line 7, the latter incrementally contains all the pixels from the top of the grid to the jth row. Assume now that, until the $(i-1)$th column, H_{col} contains valid THs. At the beginning of the processing of the ith column, H_{tmp} is cleared and, each time j is incremented, H_{tmp} is updated by adding precisely the image pixels that it lacked to be the histogram of all the points between columns $i-1$ and i of the grid from the top up to the jth row. Therefore, adding H_{tmp} to $H_{col}[j]$ as done in line 9, the latter actually contains a valid histogram of the area above and to the left of $G[i,j]$. Hence the property also holds on column i.

Therefore, at each step H_{col} contains valid THs. As lines 10–11 save those corresponding to blue and red points, i.e., to the corners of regions $R_1, \ldots, R_n$, set TH returned by Algorithm 2 contains valid THs. Applying Eq. (2) on them thus produces the same result as IH. □

Proposition 3. *Algorithm 2 never keeps in memory more than* $(4n+1)$ *THs. For a* $w \times h$ *grid, a* $N \times M$ *ROI and* B *bins, the time complexity of MSIH is* $O(n \log n + whB + NM)$ *whereas that of IH is in* $O(NMB)$.

Proof. Concerning the number of THs used by Algo. 2, remark that whenever a row has no more nonempty point, line 13 removes the TH stored in $H_{col}[j]$. Thus, there are never more THs in vector H_{col} than the number of columns where there exist nonempty points. By construction of the grid, each green square and blue triangle have at least one red point on their right. Hence, the number of THs in H_{col} is never higher than the number of red points, i.e., the number of right corners, still to be examined by the algorithm. As there are at most $4n$ corners in $R_1, \ldots, R_n$, the algorithm never uses more than $(4n+1)$ THs (including H_{tmp}).

For the time complexity, determining the rows and columns of the grid can be done in $O(4n \log 2n)$ by sorting the set of X and Y-coordinates of the corners of regions R_i. For Algo. 1, the initialization of the grid is made in $O(w \times h)$ as well as the execution of lines 4–9. Parsing once the set of R_i's, we can determine in $O(n)$ all the sets $\mathcal{R}$ and $\mathcal{L}$ that will be used on lines 10 and 15. As the union of all those sets correspond to the set of corners of the R_k's, the overall complexity of lines 10–19 is in $O(4n)$. Except the parsing of the image on line 7, there are w histograms clearings (line 4), hence a complexity of $O(wB)$. In addition, line 9 is executed at most wh times, hence a complexity of whB, and line 11 is executed at most $4n$ times. Finally, as the image pixels are parsed only once, if the ROI has size $N \times M$, then all the executions of line 7–8 involve a complexity of $N \times M$. Overall, the construction of all the THs is in $O(n \log n + whB + NM)$. ☐

4 Experimental Results

In this section, we compare three approaches in terms of computation times and memory requirements: the classical histogram approach (CH) consisting of computing the goal histograms by scanning all the pixels of their regions, the Integral Histogram (IH) which computes one TH per pixel of the ROI (see Section 2), and our approach, the Min-Space Integral Histogram (MSIH), which was described in Section 3. For this purpose, n centers of regions R_i are randomly generated using a normal distribution centered on some location (x, y) with a covariance matrix $\begin{pmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{pmatrix}$. The size $w_i \times h_i$ of the regions can also vary, but this does not play an important role in our tests. Histograms are computed on HSV images, and their number of bins B is equal to $(B_H \times B_S \times B_V)$. To make them easier to read, all the curves presenting the results of the experiments are Y-logscaled, and correspond to averages over 20 runs. The response times of IH and MSIH include the computation of the THs and the goal histograms (GH).

4.1 Computations Times

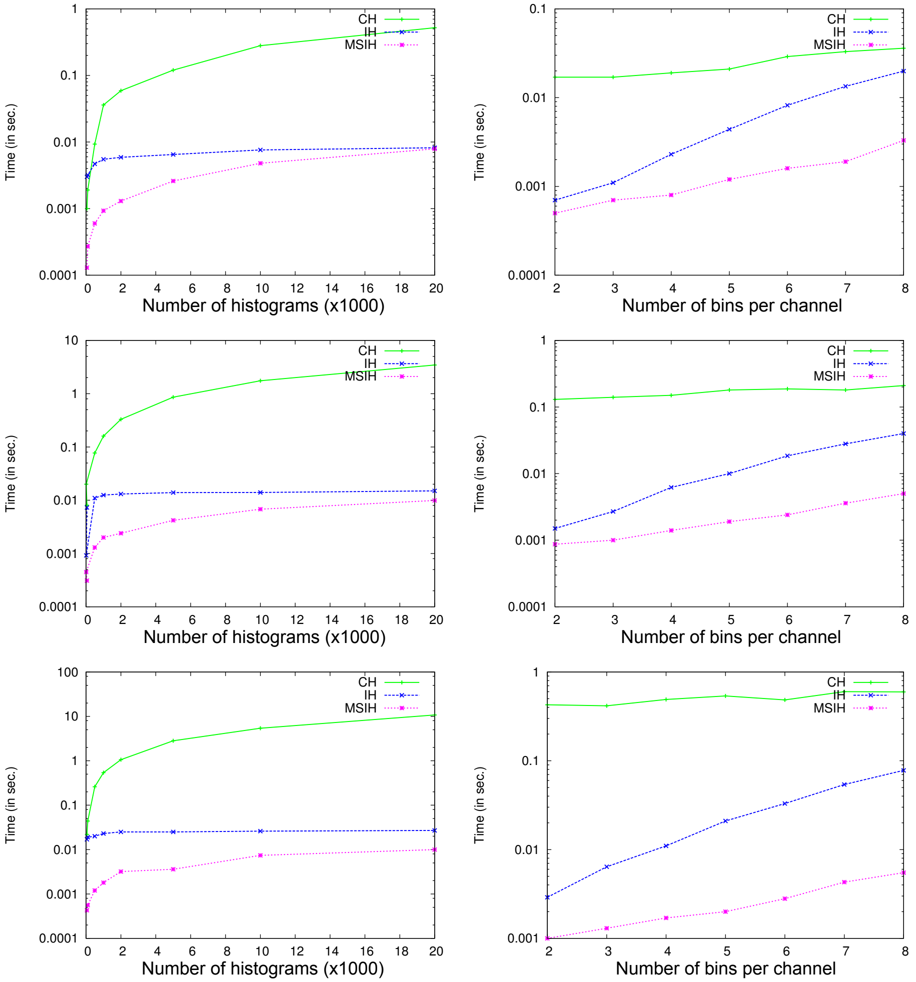
Number of computed histograms. Fig. 3.(a) reports the computation times of the $(6 \times 6 \times 4)$-bin histograms of regions R_i of size 10×10, 50×50 and 100×100 respectively, in function of the number n of regions (n are 1000 times the numbers indicated on the X-axis). As may be expected, CH linearly depends on n. For small regions, e.g., on the left graph, and for small amounts of GHs (e.g., $n < 50$), CH clearly outperforms IH because the computation of its THs is too time expensive. In all benchmarks, MSIH proved to be the fastest method, especially when regions R_i are large. MSIH outperforms IH even when numerous GHs need be computed, which corresponds to situations where IH is known to be very effective. In all our tests, MSIH is 1.5 (small regions, $n = 20000$) to 40 (large regions, $n = 50$) times faster than IH.

Quantization of histograms. One of the major drawbacks of IH is its sensitivity to the quantization of histograms: the computation of the THs (one per pixel) requires a lot of operations (and memory) when B increases. Fig. 3.(b) shows comparative results for the computation of the histograms of $n = 1000$ regions R_i of sizes 10×10, 50×50 and 100×100 respectively, in function of the number of bins per channel. To simplify, we suppose here that all channels are identically quantized. CH requires the highest computation times, and our MSIH the lowest ones. For IH and MSIH, computation times increase with B, but MSIH is less affected than IH. In all our tests, MSIH is 1.5 (small regions, $B = 2 \times 2 \times 2$) to 20 (medium regions, $B = 8 \times 8 \times 8$) times faster than IH.

Percentage of region overlap. The percentage of region overlap reflects the spatial dispersion of regions: the more spatially dispersed the regions, the smallest the overlap. In our tests, σ_x and σ_y control this percentage. This one is defined by $\%_{\text{overlap}} = 100 \times \frac{N_\cap}{N_T}$, where $N_\cap$ and N_T are the number of pixels belonging to at least two regions and belonging to at least one region respectively. Fig. 4 displays the times to compute $n \in \{100, 500, 1000\}$ $(6 \times 6 \times 4)$-bin histograms of 30×30 regions. CH is independent of the percentage of overlapping because all pixels are scanned to populate its histograms. For both IH and MSIH, computation times decrease when $\%_{\text{overlap}}$ increases because the size of the ROI in which they work decrease as well. For instance, in our tests with $n = 1000$ GHs, when 20% of the regions are overlapping, the average size of the ROI is 2800×2800 whereas it is only 30×30 for a 100% overlapping: this explains the discrepancy between the corresponding response times. Note however that MSIH is from 6 ($n = 100$, 100% overlapping) to 73 ($n = 100$, 20% overlapping) times faster than IH. This highlights the fact that IH is much more sensitive to the size of the ROI than our method: CH even outperforms IH when there is less than 90% of overlapping whereas it outperforms MSIH only when there is less than 20% of overlapping, which seldom happens in practice. Note that the profiles of the curves remain unchanged when the size of regions varies (even from very small regions to large ones).

4.2 Memory Requirements

The memory consumption of IH and MSIH depends linearly on the numbers of THs they store, which are denoted as $\#\text{histo}_{\text{IH}}$ and $\#\text{histo}_{\text{MSIH}}$ (blue and red points) respectively. Actually, each histogram is simply a B-length vector. Thus, the memory gain of running MSIH instead of IH, i.e., the percentage of



(a) (b)

Fig. 3. Computation times (logscale) for CH, IH and MSIH depending on: (a) the number n (multiples of 1000) of regions R_i of sizes $\{10 \times 10, 50 \times 50, 100 \times 100\}$ from top to bottom; GHs are $(6 \times 6 \times 4)$-bin histograms; (b) the number of bins per channels (3 channels); $n = 1000$ regions of sizes $\{10 \times 10, 50 \times 50, 100 \times 100\}$ from top to bottom.

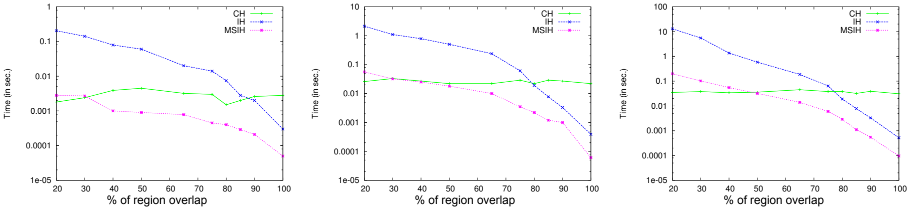


Fig. 4. Computation times (logscale) for CH, IH and MSIH depending on the percentage of region overlap, with $(6 \times 6 \times 4)$-bin histograms and regions of size 30×30, and $n = \{100, 500, 1000\}$ from left to right.

Table 1. Comparison of memory requirements for IH and MSIH to compute $(6 \times 6 \times 4)$-bin histograms of regions (overlapping percentage: 95%), depending on n and $w_i \times h_i$.

	$n =$	100	500	1000	5000	20000
small region 10×10	size of ROI	76×79	89×89	93×95	105×105	112×111
	$\#\text{histo}_{\text{IH}}$	6004	7921	8835	11025	12432
	$\#\text{histo}_{\text{MSIH}}$	376	1510	2400	4548	6137
	Gain	94%	81%	73%	59%	51%
medium region 50×50	size of ROI	117×117	128×128	134×135	144×144	150×151
	$\#\text{histo}_{\text{IH}}$	13689	16384	18090	20736	22650
	$\#\text{histo}_{\text{MSIH}}$	390	1720	3018	7593	11742
	Gain	98%	90%	84%	64%	49%
large region 100×100	size of ROI	166×167	179×178	183×184	194×194	238×237
	$\#\text{histo}_{\text{IH}}$	27722	31862	33672	37636	56406
	$\#\text{histo}_{\text{MSIH}}$	387	1723	3017	7641	19568
	Gain	99%	95%	92%	80%	66%

IH's THs that MSIH does not need to store, is given by $100 \times \left(1 - \frac{\#\text{histo}_{\text{MSIH}}}{\#\text{histo}_{\text{IH}}}\right)$. Table 1 highlights how this gain is related to the number n and to the size of the regions R_i. Observe that MSIH always computes fewer THs than IH. This is especially true when large regions are considered and n is small (MSIH's gain can rise up to 99%). This is due to the fact that, when regions become large, the size of the ROI increases, which increases accordingly the number of histograms stored by IH. Table 2 highlights the impact of the percentage of region overlap, for different values of n. A first remark concerns the fact that, for fixed values of n, $\#\text{histo}_{\text{MSIH}}$ is not very dependent on $\%_{\text{overlap}}$. On the contrary, $\#\text{histo}_{\text{IH}}$ drastically increases when this percentage decreases because the size of the ROI increases. In average, MSIH stores 1500 times fewer histograms than IH for $\%_{\text{overlap}} = 30\%$ and for any value of n. But when $\%_{\text{overlap}} = 90\%$, MSIH stores only 10 times fewer histograms.

5 Conclusion

We have introduced a new approach for fast multiple histogram computation that significantly reduces response times as well as memory consumptions, compared to both the classical approach and the well-known Integral Histogram.

Table 2. Comparison of memory requirements for IH and MSIH to compute $(6 \times 6 \times 4)$-bin histograms of regions of size 30×30 depending on the overlapping percentage and the number n of regions R_i.

	% =	30	50	70	90
$n = 100$	size of ROI	740×722	465×433	206×197	77×75
	#histo$_{\mathrm{IH}}$	534280	201345	40582	5775
	#histo$_{\mathrm{MSIH}}$	400	399	397	374
	Gain	99.9%	99.9%	99.1%	94%
$n = 500$	size of ROI	1899×1964	1287×1211	629×654	106×109
	#histo$_{\mathrm{IH}}$	3729636	1558557	411366	11554
	#histo$_{\mathrm{MSIH}}$	1999	1996	1986	1701
	Gain	99.9%	99.9%	99.6%	86%
$n = 1000$	size of ROI	2571×2711	1704×1627	713×714	171×180
	#histo$_{\mathrm{IH}}$	6969981	2772408	509082	30780
	#histo$_{\mathrm{MSIH}}$	3996	3989	3940	3385
	Gain	99.9%	99.9%	99.3%	90%

The idea relies on the fact that the number of temporary histograms that are computed by Integral Histograms can be reduced (up to 1500 times less). This induces a significant decrease of the computation times (up to 75 times less) as well as of memory requirement. Our current works concern the generalization of our algorithm to the computation of histograms of rotated regions.

References

1. Tang, G., Yang, G., Huang, T.: A fast two-dimensional median filtering algorithm. IEEE Transactions on Acoustics, Speech and Signal Processing **27** (1979) 13–18
2. Perreault, S., Hebert, P.: Median filtering in constant time. IEEE Transactions on Image Processing **16** (2007) 2389–2394
3. Sizintsev, M., Derpanis, K., Hogue, A.: Histogram-based search: A comparative study. In: proc. of CVPR. (2008) 1–8
4. Porikli, F.: Integral histogram: A fast way to extract histograms in Cartesian spaces. In: proc. of CVPR. (2005) 829–836
5. Adam, A., Rivlin, E., Shimshoni, I.: Robust fragments-based tracking using the integral histogram. In: proc. of CVPR. (2006) 798–805
6. Dubuisson, S.: Tree-structured image difference for fast histogram and distance between histograms computation. Pattern Recognition Letters **32** (2011) 411–422