Reducing images to sets of colored segments: A real-time algorithm with applications in tracking and interpretation in structured environments

Abstract

We describe an algorithm to compute a set of segments characterized with color information from an input image. The method is reversible, in the sense that an approximate reconstruction of the original image from the segments is possible. Therefore, it can be considered as a lossy compression method, useful for scenes where segments are the main primitives, such as interior of buildings, man-made objects, and so on. The compression factor is interesting by its own –approximately two orders of magnitude with respect to the original image, depending on the structure and detail of the scene–, but perhaps the main advantages of the algorithm are other: First, the compressed data can be very useful for further processing, as they are based on a simple though powerful geometric primitive, the segment with color information along each of its two sides. Second, the algorithm works in real time with low-cost hardware, such as an off-the-self image acquisition card and a standard PC. We show examples of segments extraction and reconstruction on some input images, discussing compression factors and execution times. Finally, we propose possible applications of the algorithm.

1. Introduction

Segments are powerful medium level primitives commonly used in machine vision systems where subsequent postprocessing consists in 3D reconstruction of the scene [8], or the identification of patterns in it, perhaps using a database of known objects [1]. These problems have been active research topics in the last years, mostly due to significant advances in the application of projective geometry to computer vision [4].

Projective geometry uses points and lines as its main primitives. Segments and interesting points (also known as *features*) are easily found in structured environments. Furthermore, although there exist many methods to find corners in an image, they can also be easily localized by crossing two segments whose extremes are near enough.

Color information, on the other hand, has also been successfully used in several vision problems, such as segmentation [2], tracking [6], or learning and classification [3]. But these approximations usually discard any geometric information, often working only with unstructured sets of spatially related pixels.

In this paper, we propose an efficient algorithm to reduce images to sets of colored segments, trying to take advantage of both the color information and geometric structure of the scene. We justify that the obtained output data effectively summarize the contents of the input image, as using them we can recover a very approximate reconstruction of the original image. Finally, given the efficiency of the algorithm, we propose several applications of the algorithm in systems with real-time requirements, such as robot navigation and environment reconstruction and interpretation.

2. Segment Detection

Finding lines or segments in an image is a basic problem studied in many generic computer vision references. Most of the available methods are different variations of the well known Hough Transform [5]. Other authors, on the contrary, prefer to find segments by locally grouping edge pixels into contours (for example, using the Canny operator, and then approximating them by piecewise linear polygonal approximations [4]).

In fact, the procedure used to get the segments is irrelevant to our algorithm. The only requisite is that it must be fast, as it is intended to be used in real-time applications. In our case, we use a method based on an inexpensive high-pass filtering of the image, followed by an efficient sequential grouping stage, that takes into account the local orientation of the edges. The method cannot be explained here because of the space limitations. Suffice to say that it is implemented using the Intel Image Processing Library (optimized to run in Pentium processors), and that it works at 10-20 fps (depending on the complexity of the scene) for 288×384 indoor images running on a 533 MHz CPU. We show some examples of the results obtained with the algorithm in sections 5 and 6.

3. Characterizing Segments with Color

Once the segments of the scene have been found, the next step is to label them with robust color information. We use the median gray value (or RGB, in color images) of image pixels in both sides of the segment. Figure 1 shows an algorithm that generates a table to run orderly through the environment of a given input segment. It is based on Bressenham's algorithm, a standard computer graphics method for drawing lines on pixel grids. We extend this algorithm to "thicken" the segment, by replicating the pixels in both sides of it. Observe that the algorithm uses only integer operations, thus being very fast.

Tables computed with this algorithm have two advantages: First, they cover the segment environment in a *dense* way, that is, without leaving uncovered pixels. Second, they arrange the pixels in the environment in an ordered way, by means of a rectangular array of positions (Figure 2 shows the table generated for a given example segment).

Having computed these tables, it is easy to find the median value of gray for the left side of each segment:

$$I_{med}^{left} = median(\{I_{t_{i,j}^y, t_{i,j}^x}\}_{\substack{i=0\ldots\lfloor\frac{m}{2}\rfloor-1 \\ j=0\ldots n-1}})$$

Where $I_{y,x}$ is the gray value at pixel position (y, x). The right side value can be obtained analogously using $i = \lfloor\frac{m}{2}\rfloor + 1 \ldots m-1$. The median gray value of each side can be efficiently computed, for example, by accumulating the pixel values in a histogram table, and then running through it from left to right, adding the histogram values until the cumulative sum exceeds $(\lfloor\frac{m}{2}\rfloor n)/2$ (half the total number of pixels for each side). The value at which this number is reached is the expected median value of the distribution. Finally, extension of the procedure for RGB images is trivial: we only have to apply the former equations independently to the red, green and blue channels of the image.

Input: *(Segment extremes and desired width)*
$(y_1, x_1), (y_2, x_2), W$.
Output: *($m \times n$ table of image positions)*
$(t_{i,j}^y, t_{i,j}^x)$, for $i = 0, \ldots, m-1, j = 0, \ldots, n-1$.

Algorithm:
Initialization: *(Rounding, increments and table dimensions)*
$y_1 := [y_1]$; $x_1 := [x_1]$; $y_2 := [y_2]$; $x_2 := [x_2]$;
$dy := |y_2 - y_1|$; $dx := |x_2 - x_1|$;
$m := 2\lceil W\frac{\sqrt{dy^2+dx^2}}{2(dx+1)}\rceil + 1$; $n := dx + 1$;
Main segment: *(Bressenham's algorithm)*
$e := 2dy - dx$; $y := y_1$; $x = x_1$;
for $j := 0$ ***to*** $dx - 1$ ***do begin***
$t^y_{\lfloor\frac{m}{2}\rfloor,j} = y$; $t^x_{\lfloor\frac{m}{2}\rfloor,j} = x$;
if $e \geq 0$ ***then begin***
$y = y + 1$; $e = e - 2dx$;
end
$x = x + 1$; $e = e + 2dy$;
end
$t^y_{\lfloor\frac{m}{2}\rfloor,dx} = y_2$; $t^x_{\lfloor\frac{m}{2}\rfloor,dx} = x_2$;
Segment environment: *(Segment replication)*
for $i := \lfloor\frac{m}{2}\rfloor + 1$ ***to*** $m - 1$ ***do begin***
$p_1 := (y_2 - y_1, x_2 - x_1) \cdot (t^y_{i-1,0} - y_1 + 1, t^x_{i-1,0} - x_1)$;
$p_2 := (y_2 - y_1, x_2 - x_1) \cdot (t^y_{i-1,dx-1} - y_2 + 1, t^x_{i-1,dx-1} - x_2)$;
if $|p_1| \leq |p_2|$ ***then begin*** */* Complete segment replication: */*
for $j := 0$ ***to*** dx ***do begin***
$t^y_{i,j} = t^y_{i-1,j} + 1$; $t^x_{i,j} = t^x_{i-1,j}$; */* Right. */*
$t^y_{m-i-1,j} = t^y_{m-i,j} - 1$; $t^x_{m-i-1,j} = t^x_{m-i,j}$; */* Left.*/*
end
end else begin
/ Partial segment replication, plus additional point: */*
for $j := 1$ ***to*** dx ***do begin*** */* Right: */*
$t^y_{i-1,j-1} = t^y_{i-1,j-1} + 1$; $t^x_{i,j} = t^x_{i-1,j-1}$;
end
$t^y_{i,0} = t^y_{i-1,dx-1} - y_2 + y_1 + 1$; $t^x_{i,0} = t^x_{i-1,dx-1} - x_2 + x_1$;
for $j := 0$ ***to*** $dx - 1$ ***do begin*** */* Left: */*
$t^y_{m-i-1,j} = t^y_{m-i,j+1} - 1$; $t^x_{m-i-1,j} = t^x_{m-i,j+1}$;
end
$t^y_{m-i-1,dx} = t^y_{m-i-1,0} + y_2 - y_1$;
$t^x_{m-i-1,dx} = t^x_{m-i-1,0} + x_2 - x_1$;
end
end

Figure 1. Algorithm that generates a table to run through the environment of a line. The algorithm works only in the first octant (X dominates Y, $dx > 0$ and $dy > 0$). Extensions for the rest of octants should be straightforward.

4. Image reconstruction

The procedure that recovers the original image from the set of colored segments also uses the tables described in section 3. The complete algorithm is summarized in figure 3 (for the most general case of RGB images). The pixels located immediately to the left and right of each segment are initialized and fixed with their respective RGB median values. The rest of pixels are uninitialized. Then, a simple iterative diffusion procedure is performed, in which in each iteration a RGB pixel value is recomputed as the mean of its initialized neighbors in a 3×3 window. The algorithm keeps on running until a desired convergence level is attained (typically 200-400 iterations, depending on the number of segments and the desired precision).

5. Results

Figure 4 shows the results of the segment extraction and posterior reconstruction procedure in a typical indoor scene. For this 288×384 RGB image, the computing time needed to extract the segments and get the color information was only 0.07 seconds in a Pentium III 533 MHz processor, with a Matrox frame-grabber. That means around 15 fps. The original, uncompressed image size is $288 \times 384 \times 3$=324 KB (using one byte per pixel for each red, green and blue channel). The compressed information is formed by 350 segments, each labeled with two RGB values. Each segment extreme can be coded using $\lceil \log_2(288 \times 384) \rceil = 17$ bits, and each RGB value using 24 bits. Therefore, the whole set of segments will use $350 \times ((2 \times 17) + 2 \times 24) = 28700$ bits

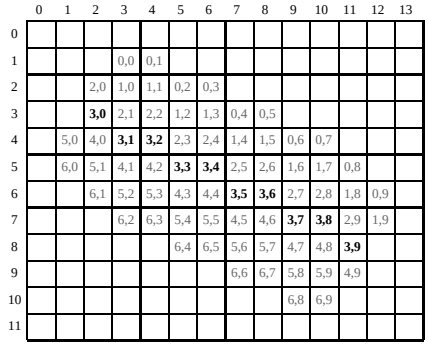

Figure 2. Pixels table for segment $((y_1, x_1), (y_2, x_2)) = ((3, 2), (8, 11))$, and width $W = 2.5$. Pair (i, j) in position (y, x) means that $(t^y_{i,j}, t^x_{i,j}) = (y, x)$.

$= 3.5$ KB. That makes a 1.1% compression factor. And the reconstructed image, though lossy, is qualitatively equivalent to the original one, at least in the geometric and color information that they contain. Thus, for virtually any subsequent post-processing, we can reliably use only the compressed information.

As we have seen, the segment extraction stage is very efficient, and works in real time. The reconstruction algorithm, on the contrary, is based on a heavy procedure that must perform a few hundreds of iterations in order to get an acceptable result image. To reconstruct the image of figure 4, for example, approximately 20 seconds were needed on the same CPU. But input image reconstruction is explicitly needed by almost no application. It is shown here mainly to illustrate the compression capacity of the segment extraction procedure. Anyway, each individual iteration is very fast (about 0.05 seconds), and the need for a great number of iterations is mostly due to the lack of a good initial estimation. But for sequences of images, the previous result can be used as the initial estimation for the next frame, and then much less iterations can attain an acceptable result. In this case, the only needed modification would be marking every pixel in the image as initialized with the values of the former reconstructed frame, fixing the pixels corresponding to the new set of segments with their new RGB values, and then iterating just as before.

Input: *(Set of S segments with colour information)*
$(y^s_1, x^s_1), (y^s_2, x^s_2), (r^s_r, g^s_r, b^s_r), (r^s_l, g^s_l, b^s_l)$, for $s = 1, \ldots, S$
Output: *(Reconstructed $h \times w$ image)*
$(I^r_{y,x}, I^g_{y,x}, I^b_{y,x})$, for $y = 0, \ldots, h-1$, for $x = 0, \ldots, w-1$.

Algorithm:
Initialization: *(Image initialization and determination of fixed points)*
for $y := -1$ *to* h *do begin*
for $x := -1$ *to* w *do begin*
$F_{y,x} :=$ Untouched; */* Flag matrix. */*
end
end
for $s := 1$ *to* S *do begin* */* For each segment s: */*
Generate table $(t^y_{i,j}, t^x_{i,j})^s$ for segment s using algorithm of figure 1, setting $W = \frac{1}{2}$ to get $m = 3$ ($i = 0, 1, 2$, and $j = 0, \ldots, n-1$);
for $j := 0$ *to* $n-1$ *do begin*
/ Left: */*
$y := t^y_{0,j}$; $x := t^x_{0,j}$;
if $0 \le y \le h-1$ *and* $0 \le x \le w-1$ *then begin*
$(I^r_{y,x}, I^g_{y,x}, I^b_{y,x}) = (r^s_l, g^s_l, b^s_l)$;
$F_{y,x} :=$ Fixed;
end
/ Right: */*
$y := t^y_{2,j}$; $x := t^x_{2,j}$
if $0 \le y \le h-1$ *and* $0 \le x \le w-1$ *then begin*
$(I^r_{y,x}, I^g_{y,x}, I^b_{y,x}) = (r^s_r, g^s_r, b^s_r)$;
$F_{y,x} :=$ Fixed;
end
end
end
Main iteration: *(Diffusion procedure)*
while not convergence *do begin*
/ Each pixel is assigned the mean of its initialized neighbours, but only if it is not fixed: */*
for $y := 0$ *to* $h-1$ *do begin*
for $x := 0$ *to* $w-1$ *do begin*
if $F_{y,x} \neq Fixed$ *then begin*

$$(J^r_{y,x}, J^g_{y,x}, J^b_{y,x}) := \frac{\sum_{k=y-1}^{k=y+1} \sum_{l=x-1}^{l=x+1} (I^r_{k,l}, I^g_{k,l}, I^b_{k,l})}{\sum_{\substack{k=y-1 \\ F_{k,l} \neq Untouched}}^{k=y+1} \sum_{\substack{l=x-1 \\ F_{k,l} \neq Untouched}}^{l=x+1} 1};$$

$F_{y,x} :=$ Touched;
end
end
end
for $y := 0$ *to* $h-1$ *do begin*
for $x := 0$ *to* $w-1$ *do begin*
$(I^r_{y,x}, I^g_{y,x}, I^b_{y,x}) := (J^r_{y,x}, J^g_{y,x}, J^b_{y,x})$;
end
end
end

Figure 3. Algorithm that reconstructs an image from a set of segments with colour information.

6. Applications

6.1. Segment tracking

Reliable matching and tracking of segments is essential to make 3D reconstruction and interpretation. Some of the available tracking methods only take into account the position of the segments in successive images. But tracking is more robust if not only the position of the primitives (points or segments) is taken into account, but also the local environment of the feature. For isolated points, one of the best methods is to evaluate local correlation of candidate points. For segments, this kind of technique can be applied to individual points along them. For example, in [7] the epipolar restriction is used to obtain corresponding points in candidate matching segments, and a correlation measure is computed for the set of pixels paired this way. But this method needs the fundamental matrix [4] to be computed for every pair of images, usually through point correspondences, which is hard for real time processing.

An alternative is to simplify the characterization of the environment of the segment, making it adequate for faster processing, and so usable in on-line applications such as

Figure 4. (Top-down, left-right) Original image, high-pass filter, segments, reconstructed image.

robot navigation. We propose to characterize each segment only with six additional scalar values, the median RGB values of the pixels along each of its two sides. These values can be used, for example, to compute a *similarity measure* between candidate pairs of segments, that takes into account both the geometric distance (as in [1]) and the color information. Using this measure, we could make the tracking more robust, without sacrificing the speed of the system.

6.2. Segment classification

The obtained median color values could also be useful in interpretation. For example, if we search for objects with a characteristic, *a priori* known color (signals for navigation, doors, the floor, etc.), we can use the color information of each segment in order to directly associate it with real objects. Figure 6.2 shows a typical indoor scene taken by an autonomous robot with a camera mounted on it. The robot can easily classify and group the segments by looking at their color information. Currently, this system is being used by our research group in a robot that navigates avoiding obstacles, obeying signals and interpreting the local situation (corridor, corner, room, and so on) in real time.

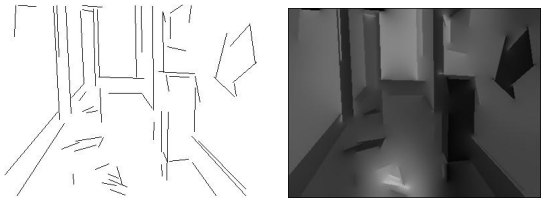

Figure 5. Indoor image taken by our robot: (Left) Segments (Right) Reconstructed image.

6.3. Image compression

The compressing capability of the algorithm can also be successfully exploited. The robot mentioned above, for example, is equipped with an onboard PC that is communicated with the rest of the world through a Radio Ethernet interface. These kind of interfaces are usually very slow (1 Mb/s bandwidth in our case). This is not enough to send images of moderate sizes at video rates of 25 fps. But, if we execute the segment extraction algorithm in the robot PC, and then send the compressed information through the network, the only bottleneck will be then the local CPU speed, not the radio interface. The rest of the processing (interpretation, color learning and classification, closed contour interpretation, calibration and 3D reconstruction) is performed in an external, unloaded computer. We have also used this technique in the application described in section 6.2.

7 Conclusions

We have described a very efficient algorithm which is able to compress images down to a 1-2% of the original size, while capturing the essential geometric structure of the scene. The algorithm works by detecting relevant segments that are then labeled with color information. The original image can be easily recovered by means of a simple diffusion algorithm. Thus, the obtained data structure contains all the relevant information of the input. This justifies the fact that the original image is no longer needed for any posterior processing. Finally, we have proposed a number of possible applications in real-time computer vision tasks, such as robot navigation and environment reconstruction and interpretation in indoor scenarios.

References

[1] J. Beveridge and E. Riseman. How easy is matching 2D line models using local search. *IEEE Trans. on PAMI*, 19(6), 1997.

[2] J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color vision for interactive robots. In *Proceedings of the IRS*, 2000.

[3] S. Buluswar and B. Draper. Color machine vision for autonomous vehicles. *International Journal for Engineering Applications of Artificial Intelligence*, 11(2), 1998.

[4] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[5] V. Leavers. Survey: Which hough transform? *CVGIP: Image Understanding*, 58:250–264, 1993.

[6] S. McKenna, Y. Raja, and S. Gong. Tracking colour objects using adaptive mixture models. *Image and Vision Computing*, 17:223–229, 1999.

[7] C. Schmid and A. Zisserman. Automatic line matching across views. In *Proceedings of the IEEE Conf. on CVPR*, 1997.

[8] Z. Zhang and O. Faugeras. A 3D world model builder with a mobile robot. *Robotics Research*, 11(4):269–285, 1992.