PUBLICATIONS DE L'INSTITUT MATHÉMATIQUE
Nouvelle série, tome 103(117) (2018), 159–173 DOI: https://doi.org/10.2298/PIM1817159N

GEOMETRICAL CODING OF COLOR IMAGES

Gleb V. Nosovskiy

ABSTRACT. Formal analysis and computer recognition of 2D color images is important branch of modern computer geometry. However, existing algorithms, although they are highly developed, are not quite satisfactory and seem to be much worse than (unknown) algorithms, which our brain uses to analyze eye information. Almost all existing algorithms omit colors and deal with grayscale transformations only. But in many cases color information is important. In this paper fundamentally new method of coding and analyzing color digital images is suggested. The main point of this method is that a full-color digital image is represented, without dropping colors, by special 2D surface in 3D space, after which it is analyzed by methods of differential geometry, rather than traditional gradient-based or Hessian-based methods (like in SIFT, GLOH, SURF, Canny operator, and many other algorithms).

1. Introduction

Nowadays pattern recognition of digital images is mainly based on limited number of algorithms which were developed during the last 40 years, such as Canny operator for edge detection, SIFT (Scale-Invariant Feature Transform), SURF (Speeded Up Robust Features), and various SIFT modifications, see [**1**–**8**]. All these algorithms use the same basic approach: 1) image is converted to grayscale; 2) grayscale image is smoothed, usually by Gaussian filters, sometimes smoothing is applied several times with increasing radius, resulting in so-called Digital Gaussian Scale-Space [**8**]; 3) using smoothed values, various types of digital gradient and Hessian characteristics are calculated at each pixel; 4) looking at these characteristics, and sometimes taking in account the situation in a vicinity of the pixel, it is decided – if this pixel is a keypoint, important for pattern recognition, or it is an unimportant point which should be excluded from further consideration; 5) for each keypoint a number of descriptors are calculated, which are used for final classification or comparison of keypoints. For example, for edge detection

2010 *Mathematics Subject Classification:* 53-04; 68T45; 68T10.

Key words and phrases: pattern recognition; geometrical coding; coding surface; contour analysis; edge detection; feature detection; motion detection; computer vision; image processing; multi-vision geometry; image stitching.

Supported by the grants NSh 7962.2016.1 N 01.200.1 17236 and 16-01-00378, provided by Russian Foundation for Basic Research.

we need to recognize keypoints which belong to some edge, for image stitching in multi-vision geometry – detect whether two keypoints on different screens reflect the same point of the same object [**9**, **10**], etc.

Although during recent decades a lot of efforts were spent on development of these algorithms, they still seem not effective enough, especially if we take into account that our brain, while processing visual information from the two eyes, does similar job much faster and better. From other hand, is very unlikely, that clumsy numerical gradient-based or Hessian-based local analysis can be really performed in our brain. It looks too artificial. This paper presents an attempt of creating more geometrical approach to the problem. The main idea is to exploit the fact that color-intensity value, situated in each pixel of a digital color image, can be uniquely represented by 3-dimensional vector. Indeed, color space is 2-dimensional and one more dimension is necessary for intensity, so the total dimension is 3. Geometrically, it means that color digital image can be represented without loss of color information by a 2D surface in 3D space. We will call such surface coding surface of the image. Note that coding surface is not unique. In order to obtain good results we should build coding surface in a proper way. Then we can use powerful machinery of modern differential geometry to analyze coding surface and get patterns from it. (We can imagine that our brain might be able to perform something like this: being a 3D body, it maybe can somehow build an "elastic" 2D surface inside and then feel tensions on it. Of course, it is only some theoretical idea, this paper does not pretend to explain how our brain works in reality, but this idea appears to be unexpectedly productive for pattern recognition.)

2. Geometrical coding of color images

2.1. Coding surface. Without loss of generality, we will assume that our digital color image is given in RGB format. RGB color space, combined with intensity axis, can be considered as the cube $\{0 \leqslant x \leqslant 1,\ 0 \leqslant y \leqslant 1,\ 0 \leqslant z \leqslant 1\}$ in R^3 see Fig. 1. Each vector in this cube represents specific color and intensity values by vector direction (color) and vector length (intensity). To make the vector length uniform with respect to colors, we may transform this cube to one eighths of the unit ball in R^3: $\{0 \leqslant x \leqslant 1,\ 0 \leqslant y \leqslant 1,\ 0 \leqslant z \leqslant 1;\ x^2 + y^2 + z^2 \leqslant 1\}$. Then length of each vector will directly represent intensity according to the scale from 0 to 1.

We will denote such cube (or ball part) as CIS (color-intensity space). Let us imagine a copy of CIS placed in each pixel of the image in such a way, that its central axis (Black-White axis) is orthogonal to the image plane, see Fig. 2.

Consider the uniquely defined vector $v \in$ CIS, which represents color and intensity values, situated in the pixel (X, Y). Then the radius vector of coding surface point, corresponding to the pixel (X, Y), is defined by the formula $\mathbf{r}(X, Y) = (X, Y) + \mathbf{v}$, see Fig. 3.

If we want blank image to be white, it is necessary to represent white color by zero vector, see Fig. 2. Otherwise, blank image will be black. In order to avoid artificial intersections of the coding surface and obtain good results, it is necessary

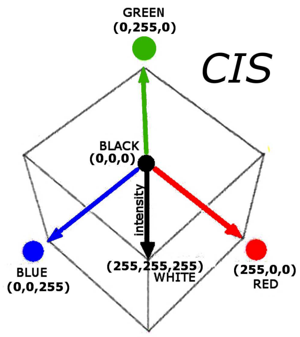

FIGURE 1. Color-intensity space (CIS) for RGB model

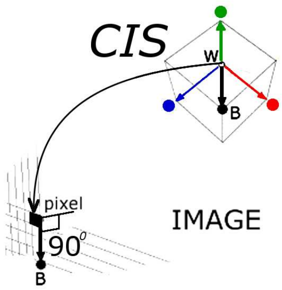

FIGURE 2. CIS cube, placed in the specific pixel of the image with its WB axis orthogonal to the image plane

to choose appropriate scale factor k_{CIS}, i.e., ratio of CIS cube size to pixel size. In our calculations we tried different k_{CIS} values from 1 to 50; usually values around 10–20 work fine. In the case of grayscale image the coding surface will coincide (up to a scale coefficient) with intensity (or inverse intensity) chart. For a plain monochrome image its coding surface is a plane rectangle, see Fig. 4.

2.2. Image smoothing by Bezier or NURBS approximation. Real color digital image usually contains considerable amount of noise, so it is necessary to smooth it before investigation. In our method smoothing is obtained automatically by replacing a local patch of coding surface by its Bezier or NURBS approximation, see example on Fig. 5. To perform smoothing, we just consider each point $r(X, Y)$ on coding surface, corresponding to image pixel (X, Y), as control point of Bezier or NURBS surface [**11**, **12**].

We used Bezier rather than NURBS approximations. The reason is that Bezier approximations are simpler, do not require setting of control parameters, such as knot vector in NURBS case, and are much more viscous than NURBS approximations, which is good for noise suppression. The rate of smoothing by a Bezier

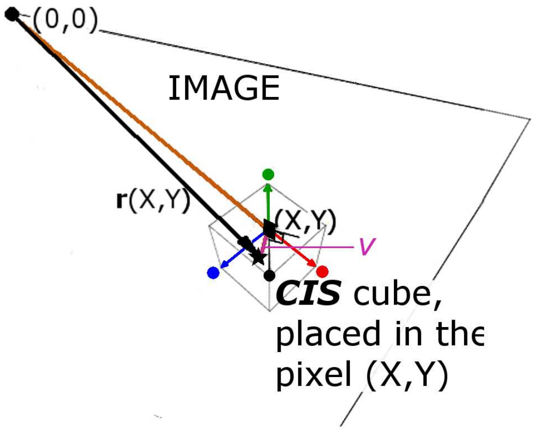

FIGURE 3. Construction of coding surface for a digital color image: $\mathbf{r}(X, Y)$ is radius vector of surface point (marked by star), corresponding to the image pixel with coordinates (X, Y)

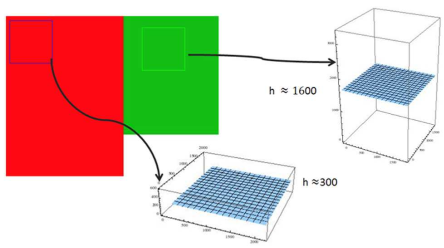

FIGURE 4. Coding surfaces for plain monochrome images are plane rectangles, typically located on different levels h (values of h are shown here in conventional units)

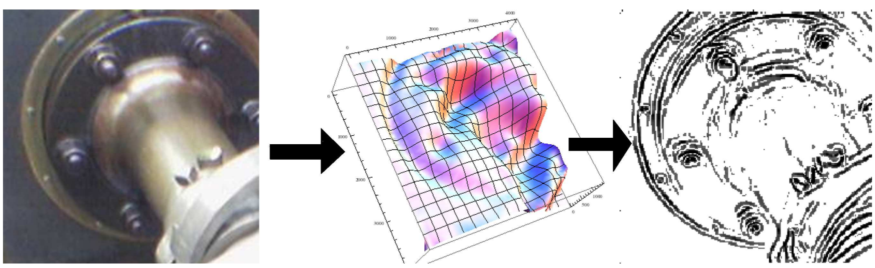

FIGURE 5. Color image → coding surface → edge detection. We use here a fragment of image from `en.wikipedia.org`

approximation is entirely defined by the number of control points, taken into account, i.e., by the size of Bezier patch (bigger the size – stronger the smoothing).

Note that for the investigation of certain vicinity of the pixel we should build Bezier approximation for the corresponding patch of coding surface, rather than considering a patch of Bezier approximation for the whole coding surface. The latter could be too over-smoothed. In our calculations we used patch size in the range $5 \times 5 - 9 \times 9$, usually 7×7.

3. Edge detection based on geometrical coding

Using coding surface, image pattern recognition could be performed in a different way. Here we will discuss one of the most important branches of image pattern recognition – edge detection.

Generally, edges are lines, which separate image areas with significantly different values of color or intensity. On coding surface such lines are reflected by zones of distinctive metric deformation, and also by creases, along which coding surface appears to be folded. Therefore, the problem of edge detection on a color image can be transferred into the problem of tension and crease detection on its coding surface. In this paper we will mainly discuss crease detection. Only few examples of tension detection will be presented here. A natural way to detect creases on a smooth surface is based on observation, that principal curvatures of a $2D$ surface in R^3 are typically very different in absolute value along creases: curvature in the direction of crease is usually rather small, and another one, in orthogonal direction, is big, see Fig. 6.

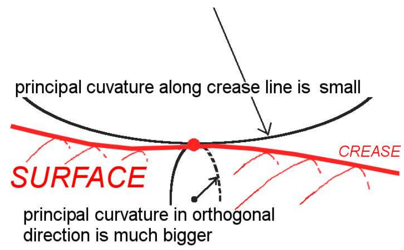

FIGURE 6. Crease detection by principle curvatures

Therefore, in order to detect creases on a coding surface, the following steps could be performed:

1) Let us choose a scalar function $f(x) = f(\lambda_1(x), \lambda_2(x)) \geqslant 0$ on a coding surface S, depending on the two principal curvatures $(\lambda_1(x), \lambda_2(x))$ at $x \in S$, in such a way, that the value of f rises when one curvature is big and another is small.

2) Consider a set of points $A \subset S$ on the coding surface S, determined by the condition: $A = \{x : f(x) > a\}$ where a is some threshold. Then, for an appropriate f and proper choice of the threshold a, the set A will form a chart of creases on the coding surface. The projection of this chart to the image will form an edge drawing.

Of course, such edge drawing will depend on the choice of function f and threshold a. With help of different f and a we can make our edge drawing more or less detailed.

Here are some examples of functions, which we used:

$$
\begin{gathered}
f_0(\lambda_1, \lambda_2) = c\big||\lambda_1| - |\lambda_2|\big|, \\
f_1(\lambda_1, \lambda_2) = c|\lambda_{\max}|/\lambda_{\min}^2, \\
f_2(\lambda_1, \lambda_2) = c(\lambda_1 - \lambda_2)^2 = c(H^2 - 4K), \\
f_3(\lambda_1, \lambda_2) = c(\lambda_1^2 - \lambda_2^2)^2 = cH^2(H^2 - 4K), \\
f_4(\lambda_1, \lambda_2) = c(|\lambda_1| - |\lambda_2|)^2 = \begin{cases} c(H^2 - 4K), & \text{if } K \geqslant 0 \\ cH^2, & \text{if } K < 0 \end{cases} \\
f_5(\lambda_1, \lambda_2) = \frac{(\lambda_1^2 - \lambda_2^2)^2}{\lambda_1^2 \times \lambda_2^2} = \frac{H^2(H^2 - 4K)}{K^2} = \frac{\lambda_1^2}{\lambda_2^2} + \frac{\lambda_2^2}{\lambda_1^2} - 2, \\
f_6(\lambda_1, \lambda_2) = \frac{(|\lambda_1| - |\lambda_2|)^2}{H^2} = \begin{cases} 1 - 4K/H^2 & \text{if } K \geqslant 0; \\ 1 & \text{if } K < 0 \end{cases} \\
f_7(\lambda_1, \lambda_2) = \frac{(\lambda_1 + \lambda_2)^3}{\lambda_1^2 \times \lambda_2^2} = \frac{H^3}{K^2} \\
h_1 = c \times \det G \quad \text{(serves for tension detection).}
\end{gathered}
$$

Here c is the scaling coefficient, G denotes the first fundamental form (Riemannian metric tensor), $H = (\lambda_1 + \lambda_2)$ – mean curvature, $K = (\lambda_1 \times \lambda_2)$ – Gaussian curvature; $\lambda_{\max}, \lambda_{\min}$ are principal curvatures of maximal and minimal absolute value.

Note that the functions $f_2, f_3, f_4, f_5, f_6, f_7$ are rational expressions from H and K, so they do not require principal curvatures calculation. Let us recall that H and K are, in their turn, rational expressions from matrix elements of first and second fundamental forms, while principal curvatures calculation requires square root extraction.

Note that the function f_7 behaves similarly to f_1. Indeed, if $\lambda_{\min}$ is much smaller in absolute value than $\lambda_{\max}$, then $\lambda_1 + \lambda_2 = \lambda_{\max} + \lambda_{\min} \approx \lambda_{\max}$ and therefore $f_7(\lambda_1, \lambda_2) \approx f_1(\lambda_1, \lambda_2)$. Calculations for f_7 are faster than for f_1, so f_7 can be used instead of f_1.

Note that the functions f_5, f_6 are homogenous functions from λ_1, λ_2, so for their calculation there is no need to normalize the length of normal vector to the surface. For a Bezier surface it means that f_5, f_6 could be written as rational expressions from the initial data (coordinates of control points).

Let us recall that matrix elements of the first and second fundamental forms for a Bezier surface are simple linear expressions from coordinates of 6 control points (3 points for the first and 3 more points for the second form) of Bezier surface patch, obtained by the division of original Bezier surface in the point of consideration [**11**, **12**].

4. Some examples or edge detection using geometrical coding

Here we present several examples of edge detection with different functions $f(\lambda_1, \lambda_2)$ and function $h_1(G)$. All examples were calculated using computer program which was written in C++ and Wolfram Mathematica by my students Alexey Chekunov and Sergey Podlipaev (Moscow Lomonosov State University, Mech.-Math. Faculty). The program calculates principal curvatures of Bezier approximation of the $d \times d$ patch of coding surface with point of consideration in the center. Then, using one of the possible functions $f(\lambda_1, \lambda_2)$ or $h(G)$, and a certain threshold a, picture of edges is obtained as the set of points: $\{x : f(x) > a\}$ or $\{x : h(G(x)) > a\}$.

Our aim here is to show principal abilities of geometrical coding approach, so all results are presented in their original form; no postprocessing for improving edges was applied.

We start with a standard example of edge detection from Wikipedia article “Canny edge detector". In Fig. 7 images a) and b) were taken from this article; images c) and d) are results of image processing by our algorithm with functions f_1 and f_0 correspondingly. Note that Canny edge detection implies post-processing for eliminating double contours and making them thinner, while our edge detection is presented here “as it is".

From Fig. 7 it is clear that creases structure on coding surface contains detailed information about edges on original image. Therefore geometrical coding approach can be really used for fast and effective edge detection.

In Fig. 8 we present rather complicated color image with mixture of sharp and blurred shapes. Here Canny algorithm works well on sharp edges, but poorly detects blurred reflections on dark pavement. At the same time, GC detection works well everywhere.

In Fig. 9 we present a general example of daylight photo. Canny edge detector works well here, but GC detection is more flexible in showing details.

In Fig. 10 another example of color photo is presented together with the results of edge detections. Here GC edge detector looks noticeably better than Canny operator.

In Fig. 11 edge detection is illustrated on a really complicated color photo with a lot of tiny details in light and shadow areas. Again GC algorithm visibly over-performs Canny operator. Note that light areas can look darker on edge chart because in light areas it is easier to detect edges, so more black lines are present there.

Geometrical coding approach works not only for color, but for grayscale images also. In Fig. 12 we show a comparison between GC edge detection, Canny operator, Sobel-Feldman operator, and several other gradient methods for grayscale image. Note that GC detection gives one of the best results here.

Our last example will be a digital color photo of a human face—challenging task for any edge recognition algorithm. Alexey Chekunov, MSU postgraduate student, who participated in creating computer program for geometrical coding, used his own photo as example. It was shot by a 12M camera mounted in a cell

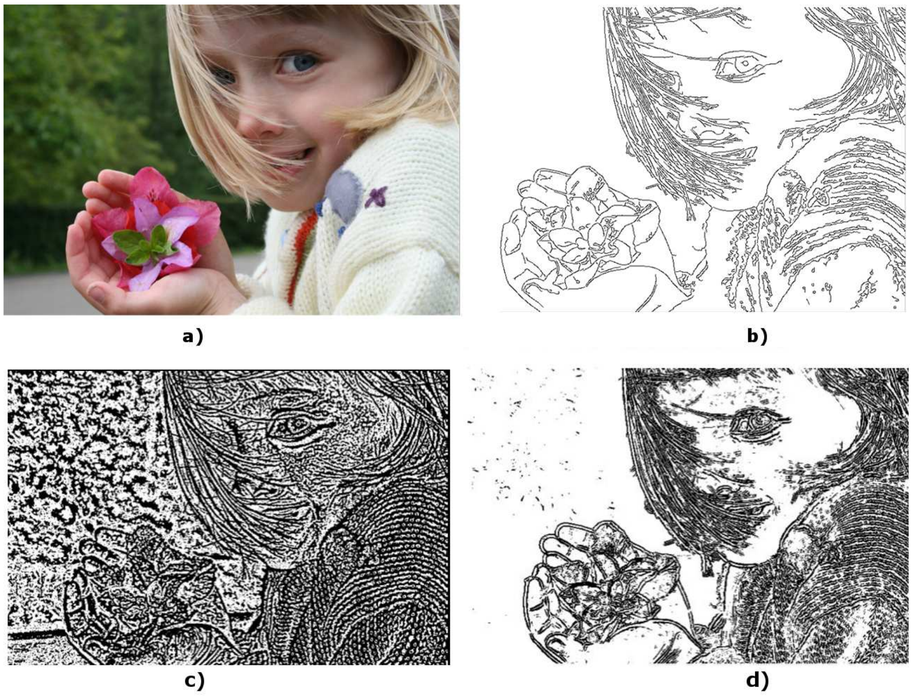

FIGURE 7. a) Original image; b) Canny edge detection; c) Geometrical coding edge detection with function f_1; d) Geometrical coding edge detection with function f_0

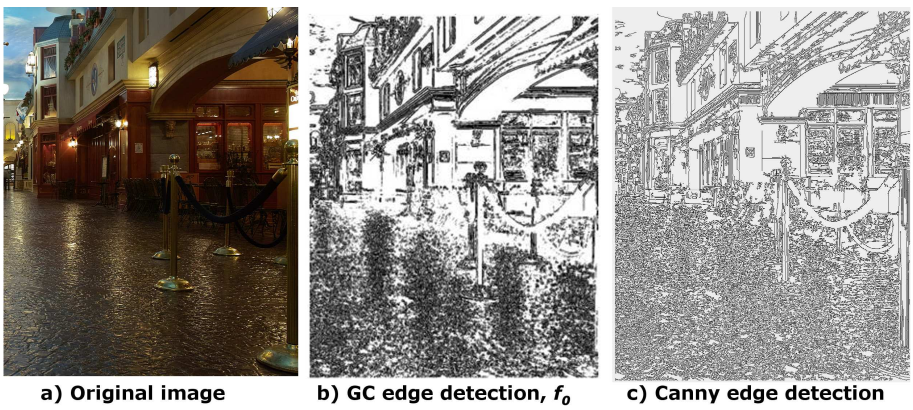

FIGURE 8. a) Original image; b) Geometrical coding edge detection with function f_0; c) Canny edge detection

phone. The results are presented in Fig. 13. Note that Canny operator practically failed here, while GC edge detection works fine.

FIGURE 9. a) Original image; b) Canny edge detection; c) GC edge detection with function f_1 ; d) GC edge detection with function f_0

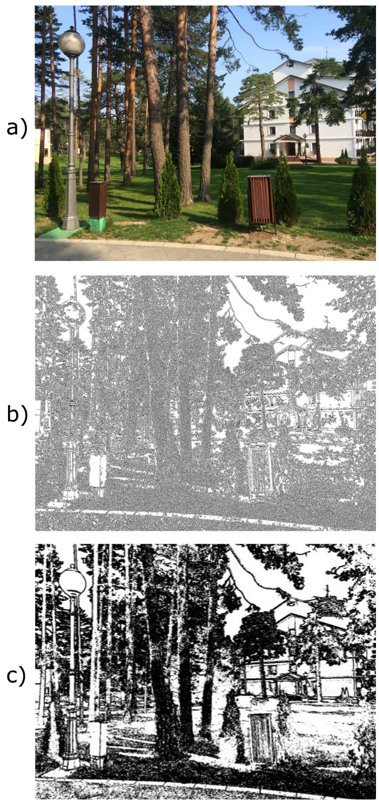

FIGURE 10. a) Original image; b) Canny edge detection; c) GC edge detection with function f_4

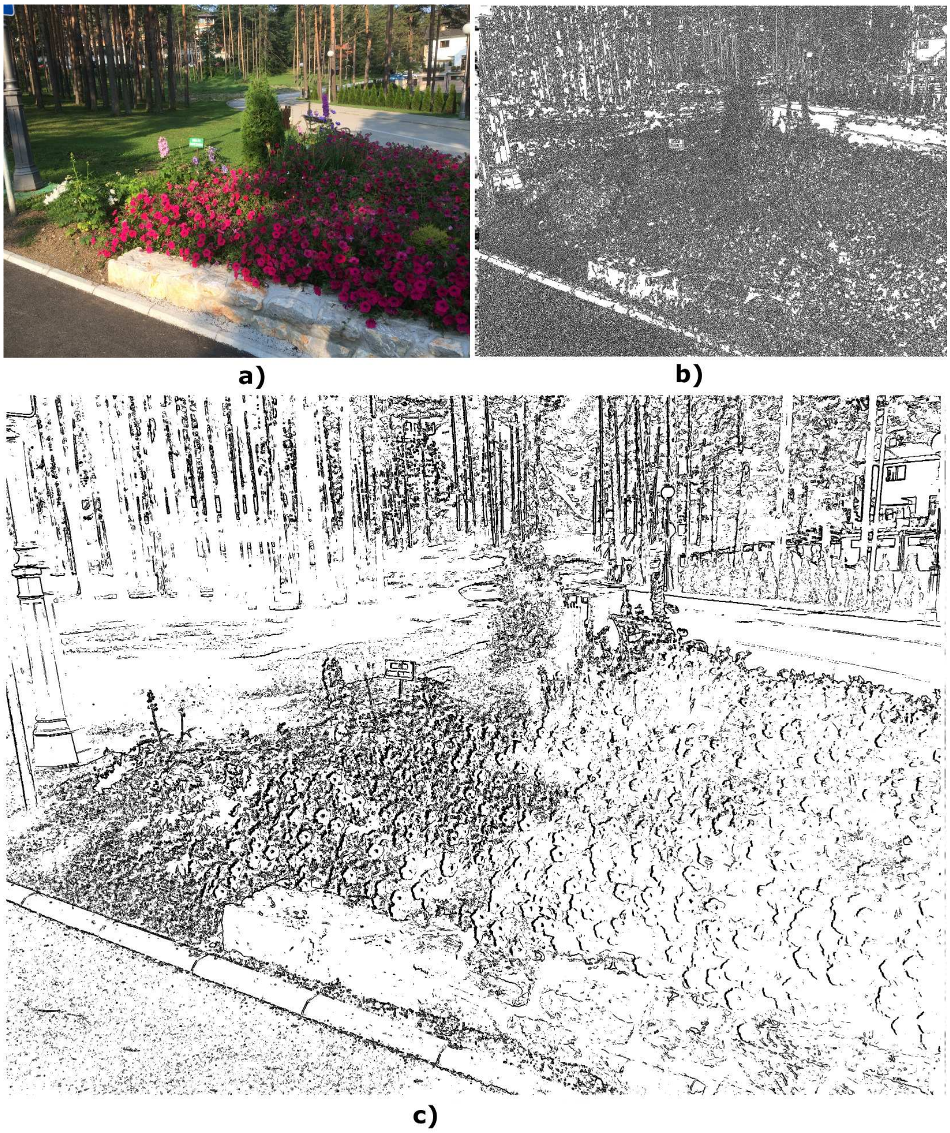

FIGURE 11. a) Original image; b) Canny edge detection; c) GC edge detection with function h_1

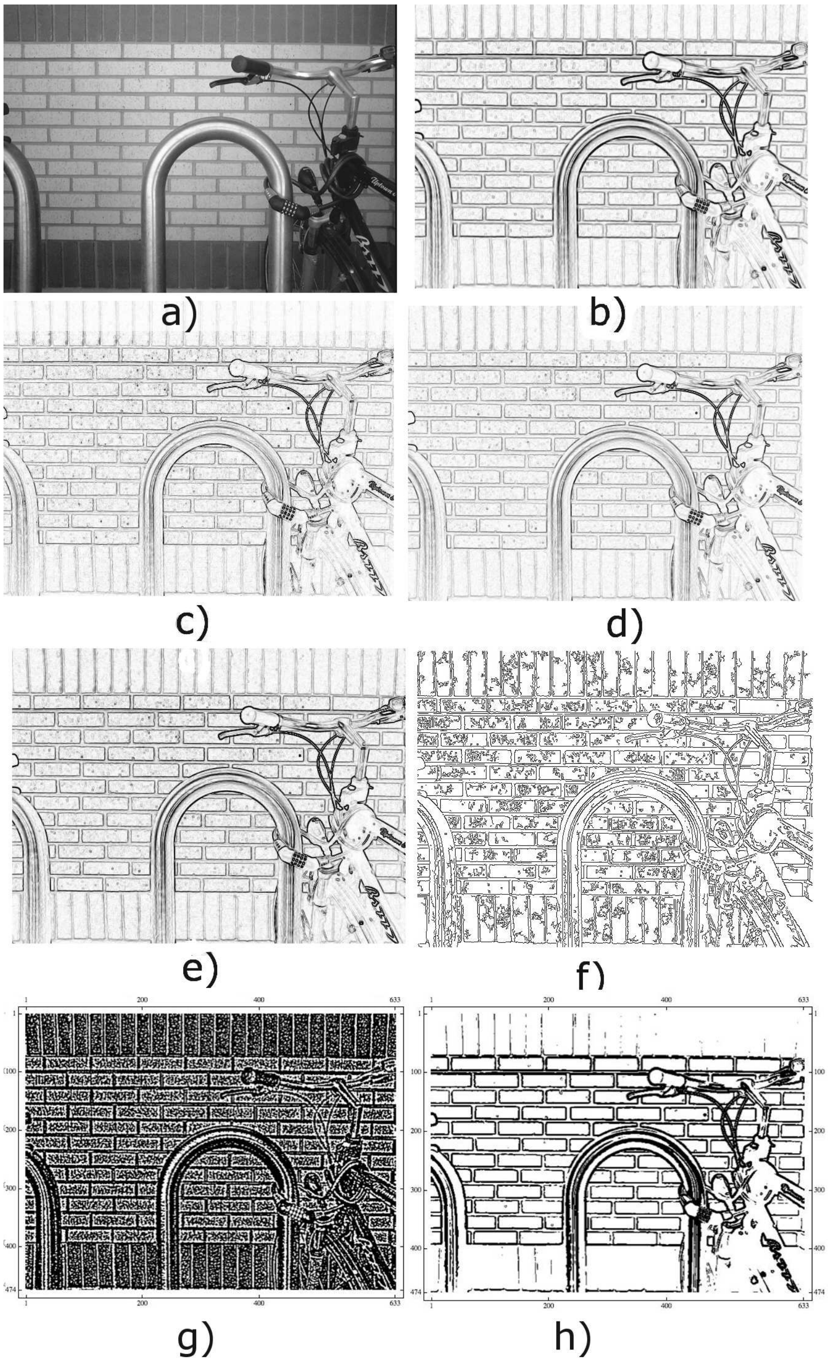

FIGURE 12. a) Original image; b) Sobel–Feldman; c) Roberts Cross; d) Scharr; e) Prewitt; f) Canny; g) GC with function f_7; h) GC with function h_1. Images a)-f) were taken in August 2016 from Wikipedia article about Sobel operator

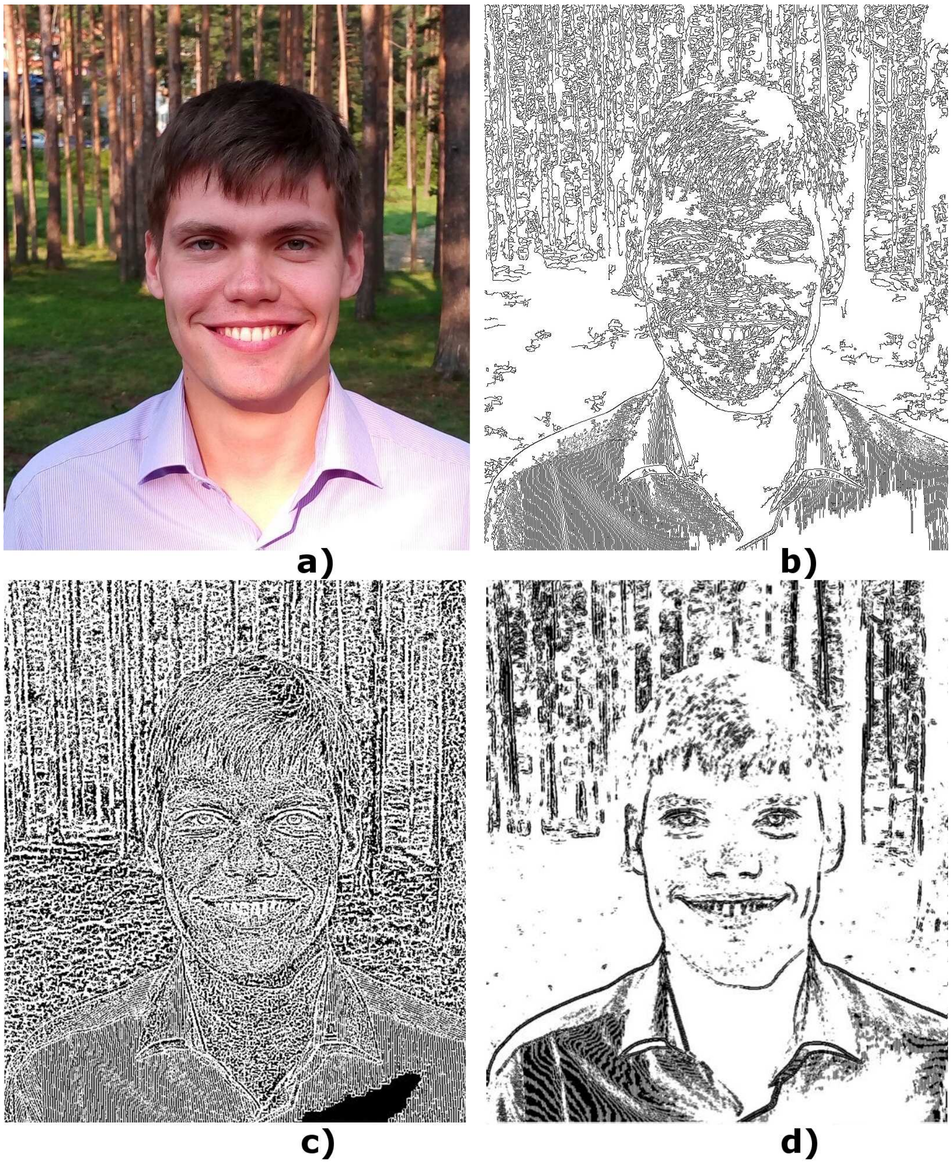

FIGURE 13. Edge detection on a color photo of human face: a) Original image; b) Canny edge detection; c) GC edge detection with function f_1 (inversed picture); d) GC edge detection with function f_4

5. Application to multiple-view geometry

GC edge detection can be implemented in multiple-view geometry algorithms for creating a stereoscopic picture from two images, obtained from two, sometimes close to each other, points of view (like human eyes). One of the main tasks here is a fast recognition of conjugate points on both images (conjugate points are such points which reflect the same point on some object in view). It could be done using preliminary determination of approximate shift between two given images [**9**, **10**, **12**, **13**]. Such determination can be performed using edge detection: at first edge contours are detected on both images, and then approximate shift between them is calculated. This procedure is illustrated in Fig. 14.

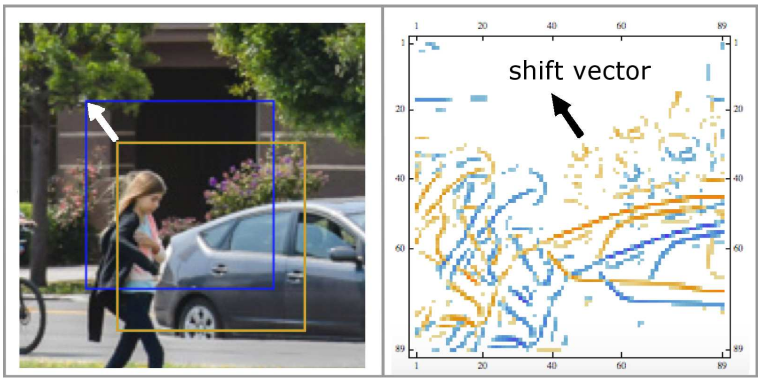

FIGURE 14. Shift calculation by means of GC edge detection. Function f_0 was used here

6. Computational complexity

Control points for the local Bezier approximation of coding surface, which are used in the further computations, are directly determined by the initial data with help of only few additions and multiplications per pixel.

Next, in the case of homogenous function $f(\lambda_1, \lambda_2)$ we need to calculate 6 matrix elements of first and second fundamental form up to an arbitrary coefficient. Both forms are given by a 2×2 symmetric matrix, so we need to calculate 3 scalar elements in each of them, totally 6 scalars. From differential geometry we know that all of them could be expressed through scalar and vector products of 5 first and second order derivatives $r_u, r_v, r_{uu}, r_{uv}, r_{vv}$ of the 3-dimensional radius vector r of the surface. Totally, we need about 40 additions and multiplications per pixel in order to calculate both fundamental forms from derivatives $r_u, r_v, r_{uu}, r_{uv}, r_{vv}$ and about 15 additions and multiplications per pixel in order to calculate only first fundamental form from derivatives r_u, r_v.

Vectors $r_u, r_v, r_{uu}, r_{uv}, r_{vv}$ are expressed through 6 corner control points

$$p_{00}, p_{01}, p_{02}, p_{10}, p_{11}, p_{20}$$

of right-upper part of local $d \times d$ Bezier surface, divided in the regarded point (which is the lower-left corner p_{00} of this part), by the formulas (see, for example, [**11**, **12**], p. 210):

$$r_u = (d-1)(p_{10} - p_{00}); \quad r_v = (d-1)(p_{01} - p_{00});$$
$$r_{uu} = (d-1)(d-2)(p_{20} - 2p_{10} + p_{00});$$
$$r_{vv} = (d-1)(d-2)(p_{02} - 2p_{01} + p_{00});$$
$$r_{uv} = (d-1)^2(p_{11} - p_{10} - p_{01} + p_{00}).$$

Let us recall that each control point $p_{ij}, 0 \leqslant i, j \leqslant (d-1)$ of the upper-right part of the divided Bezier surface of $d \times d$ size can be expressed by certain linear function from $(d-i) \times (d-j)$ control points of the original local Bezier surface the coordinates of which are directly obtained from the initial data [**11**, **12**]. It is easy to count up, that calculation of 6 required control points $p_{00}, p_{01}, p_{02}, p_{10}, p_{11}, p_{20}$ will take $d \times d + 2(d-1)d + 2(d-2)d + (d-1)(d-1) = (6d^2 - 4d - 3)$ additions and multiplications per coordinate, which gives the total amount of $(18d^2 - 12d - 9)$ additions and multiplications per pixel for all the three coordinates of all 6 required vectors p_{ij}.

The total complexity will be $18d^2 - 12d - 9 + N$ additions and multiplications per pixel. Here N is about 40, and $d \times d$ is the size of a region, used for the local Bezier approximation.

For $d = 7$, which usually is quite enough, the total amount of calculations is about 830 additions and multiplications per pixel for computation of a homogeneous $f(\lambda_1, \lambda_2)$. Similar calculations show that the function $h_1(G)$, which is based on the first fundamental form only, requires $3(d \times d + 2(d-1)d) + 15 = 9d^2 - 6d + 15$ additions and multiplications per pixel which is about 420 for $d = 7$.

If we consider the local Bezier surface with lower-left corner in the regarded point, we do not need to divide it. In this case, if we take $d = 3$, the total amount of additions and multiplications per pixel will be about 40 for homogeneous $f(\lambda_1, \lambda_2)$, and about 20 for functions, based on the first fundamental form only, such as $h_1(G)$.

In the case of nonhomogenous function $f(\lambda_1, \lambda_2)$, which can be expressed through mean and Gaussian curvatures H and K, we need additional calculation of one inverse square root per image pixel in order to normalize normal vector to the surface in regarded point.

For nonhomogenous function $f(\lambda_1, \lambda_2)$, which cannot be expressed through H and K, it is necessary to perform one more square root extraction per pixel for λ_1, λ_2 calculation.

Therefore total calculation complexity of GC edge detection is linear with respect to number of pixels, involved in calculation (sometimes it is enough to calculate not in all pixels, but in some disperse set only). Per pixel complexity is from about 40 additions and multiplications for the simplest approach with $d = 3$ to about 830 additions and multiplications for more complicated approach with $d = 7$ for functions $f(\lambda_1, \lambda_2)$. For function $h(G)$ the corresponding amounts are about 20 and 420 additions and multiplications per pixel. Some functions $f(\lambda_1, \lambda_2)$ require also l or 2 root extractions (one of which is inverse) per pixel.

7. Conclusions

Geometrical coding is new promising approach to pattern recognition on color digital images. It opens a way to analyze full-color images without converting them to grayscale. GC method has linear computational complexity with respect to image volume. Resolution of GC-based edge detection from the very beginning appears to be comparable or even better than resolution of all algorithms known today. This paper presents the basic conception of GC approach and a number of initial examples.

References

1. H. Bay, A. Ess, T. Tuytelaars, L. V. Gool, *SURF: Speeded Up Robust Features*, Comput. Vis. Image Underst. **110**(3) (2008), 346–359.
2. J. Canny, *A Computational Approach for Edge Detection*, IEEE Trans. Pattern Anal. Mach. Intell. **8**(6) (1986), 679–698.
3. D. G. Lowe, *Object recognition from local scale-invariant features*, Proc. Internat. Conf. Comput. Vision (1999), 1150–1157.
4. J. Beis, D. G. Lowe, *Shape indexing using approximate nearest-neighbor search in high-dimensional spaces*, Conf. Comput. Vision Pattern Recogn., Puerto Rico (1997), 1000–1006.
5. K. Mikolajczyk, C. Schmid, *A performance evaluation of local descriptors*, IEEE Trans. Pattern Anal. Mach. Intell. **10**(27) (2005), 1615–1630.
6. H. Bay, T. Tuytelaars, L. Van Gool, *SURF: Speeded Up Robust Features*, Proc. Ninth Eur. Conf. Comput. Vision, May 2006 (2006).
7. T. Lindeberg, *Image matching using generalized scale-space interest points*, J. Math. Imaging Vis. **52**(1) (2015), 3–36.
8. R.-O. Ives, M. Delbracio, *Anatomy of the SIFT Method*, Image Processing On Line, 2014-12-22, available at http://dx.doi.org/10.5201/ipol.2014.82, (2014).
9. G. V. Nosovskiy, *Computer Gluing of 2D Projective Images*, in: *Proc. Workshop "Contemporary Geometry and Related Topics" in Belgrade, Yougoslavia, 15–21 May 2002*, World Scientific, New Jersey, London, Singapore, 2004, 319–334.
10. R. Hartley, A. Zisserman *Multiple View Geometry in Computer Vision*, Cambridge University Press, 2000.
11. M. Hosaka, *Modelling of Curves and Surfaces in CAD/CAM*, Springer-Verlag, Berlin, Heidelberg, 1992.
12. N. N. Golovanov, A. T. Fomenko, D. P. Ilytko, G. V. Nosovskiy, *Computer Geometry*, Academia, Moscow, 2006 (in Russian).
13. G. V. Nosovskiy, E. S. Skripka, *Error Estimation for the Direct Algorithm of Projective Mapping Calculation in Multiple View Geometry*, in: *Proc. Workshop "Contemporary Geometry and Related Topics" in Belgrade, Serbia and Montenegro, June 26 – July 2, 2005*, Faculty of Mathematics, University of Belgrade, 2006, 399–408.

Faculty of Mechanics and Mathematics
Moscow Lomonosov State University
Moscow
Russia
gleb.nosovskiy@gmail.com