Proceedings of ALGORITMY 2000
Conference on Scientific Computing, pp. 174–187

AN ADAPTIVE FINITE VOLUME METHOD IN PROCESSING OF COLOR IMAGES

ZUZANA KRIVÁ* AND KAROL MIKULA†

Abstract. We study, from a computational point of view, a model for processing of RGB images based on a regularized (in the sense of Catté, Lions, Morel and Coll) Perona-Malik nonlinear image selective smoothing equation. The model is represented by a system of nonlinear partial differential equations with a common diffusion coefficient given by a synchronization of the information coming from all color channels. For the numerical solution we adjust a finite volume computational method given in [7] and propose a coarsening strategy to reduce a number of unknowns in the linear system to be solved at each discrete scale step of the method.

Key words. image processing, RGB image, nonlinear partial differential equations, numerical solution, finite volume method, adaptivity, grid coarsening

AMS subject classifications. 35K55, 65P05

1. Introduction. A RGB image can be viewed as a composition of three grey-scale images representing level of intensity for red, green and blue colors. Any of these scalar images can be modelled by a real function $u_i^0(x), i = 1,2,3$, defined in some rectangular subdomain $\Omega \subset I\!R^d$ (in practice $d = 2$ or 3). Applying an evolutionary partial differential equation (PDE) to the initial image $u_i^0(x)$ is known as the *image multiscale analysis* ([1], [3]). It associates with the initial image $u_i^0(x) = u_i(0,x)$ a family of "simplified" images $u_i(t,x)$, a solution of PDE, depending on an abstract parameter $t > 0$ called scale.

In this paper, we consider a Perona-Malik type ([10]) system of equations, regularized in the sense of Catté, Lions, Morel and Coll ([5]), which we adapt to a RGB image. In our model we will not apply Perona-Malik-like equation to each channel independently (which would be the most simple possibility), but we synchronize the diffusion in each channel by computing common diffussion coefficient depending on an information coming from all three colors (see also [13], [12] dealing with similar techniques in a vector valued diffusion and a color image processing). Thus, we consider the following system of nonlinear partial differential equations

$$\partial_t u_i - \nabla.(d\nabla u_i) = 0, \quad i = 1,2,3 \tag{1.1}$$

in $Q_T \equiv I \times \Omega$, where $I = [0,T]$ is a scaling interval and

$$d = g(\sum_{i=1}^{3} |\nabla G_\sigma * u_i|), \tag{1.2}$$

together with initial and zero Neumann boundary conditions for each channel. Let g be a smooth nonincreasing positive function with $g(0) = 1$ tending to zero at infinity, $G_\sigma \in C^\infty(I\!R^d)$ be a smoothing kernel with $\int_{I\!R^d} G_\sigma(x)dx = 1$ and let every initial function $u_i^0 \in L^2(\Omega), i = 1,2,3$. In the convolution term the kernel is applied to a periodic extension of the image u_i from Ω to $I\!R^d$.

*Department of Mathematics, STU Bratislava, Slovakia, e-mail: kriva@vox.svf.stuba.sk
†Department of Mathematics, STU Bratislava, Slovakia, e-mail: mikula@vox.svf.stuba.sk

The basic idea of Perona and Malik involves a controlling the diffusion (smoothing) of the image with help of a diffusion coefficient in the nonlinear parabolic equation by means of its dependence on ∇u which is, in a sense, an edge indicator. Catté, Lions, Morel and Coll considered $\nabla G_\sigma * u$, the *Gaussian gradient*, for decision where there is non-spurious / spurious(noisy) edge.

In the case of (1.1)-(1.2), if a non-spurious edge is present in all three channels, g returns a smaller value than in the case when the channels are processed indepedently, and thus the edge is better preserved. If a noise is present in only one of the channels, the model works in the same way as for the greyscale image. If a noise is present in all three channels at the same time, smoothing can be slower at the beginning, but with the increasing scale, the difference diminishes.

For the numerical solution of (1.1)-(1.2) we adjust a technique for a greyscale image suggested and analysed in [7]. It is based on a semi-implicit discretization in scale and on the so called finite volume method in space. Recently, the finite volume method (FVM) has been widely used in computational sciences and engineering since it is based on physical principles as conservation laws, it is local and easy to implement. Moreover, in the FVM the discrete approximations of a solution of partial differential equation are considered to be piecewise constant in control volumes (cells) which in the image processing corresponds to pixel structure of a discrete image. From conceptional point of view such approach seems to be *the most natural for the image processing.*

Since with an increasing scale a solution tends to be more flat in large regions of the image, we can improve efficiency of the method using adaptivity. It is not necessary to consider the same fine resolution in the whole spatial domain. This access reduces the computational effort, because a coarsening of the computational grid reduces the number of unknowns in the linear systems to be solved at the discrete scale steps of the method. Since all the information about the image is contained in the initial grid and there is no spatial movement of the edges, no refinement is needed and we work just with grids, elements of which are obtained by merging of pixels. This process is called *coarsening* in the numerical methods for solving PDEs. In this paper, we present coarsening strategy for rectangular grids and join such strategy with the finite volume method for (1.1)-(1.2). For the finite element method such approach has been suggested for image processing applications in [4]. The method has been based on triangular grids generated by bisection which are successively coarsened during the diffusion process. The approach given in [4] has been adjusted for bilinear finite elements by Preusser and Rumpf in [11]. They also improve storage requirements of the method by procedural handling of adaptive quadtrees benefiting from a general and efficient multilevel data post processing methodology discussed in [8],[9].

In Section 2 we describe strategy for creating the adaptive grid. In Section 3 we present the finite volume method on such grid and in Section 4 we discuss computed numerical experiments.

2. Creating of the adaptive grid. The initial image is given as a set of discrete grey (or RGB) values on cells (finite volumes) of a uniform grid. Every element of such a grid corresponds to one unknown in a resulting linear system. To decrease the number of unknowns, we can decrease number of elements: at the beginning and especially with the increasing scale, we can merge cells using some coarsening criterion and instead on the regular grid, we can work on the irregular adaptive structure. For its generation we chose an approach based on quadtrees, which are the most convenient way to produce *graded* meshes (it means,in our case, that we have small elements, where the image information changes (near the edges), and large ones, where it is of a constant mean). Moreover, the quadtree itself may be computed in integer arithmetic. A *quadtree* is a recursive partition of a region of the plane into axis-aligned squares. One square, the root, covers the entire region. By splitting it with horizontal and vertical line segments through its center, a square can be divided into four child squares-quadrants. Let the quadtree square be called a *leaf* square, if it is not subdivided into children. Then the image is represented by the leaves of the quadtree. The criterion, ruling the subdivision of the quadrants, depends on an intensity difference of pixels contained in a given square. If it is smaller than a given tolerance ε, we will continue in the subdivision, otherwise we will stop it. To simplify creating the matrix of the linear system, we require fulfilling of the *balance condition*: no quadtree leaf is adjacent to another leaf of more than twice its size.

The process of the division of quadrants can be formally written with the help of a tree (Figure 1). Various linearized descriptions with low memory requirements are used in practice. Here we use a representation based on a preorder type (or depth-first) traversing of the tree, resulting in two lists: binary list L_1 encoding the structure, and a list of image values L_2.

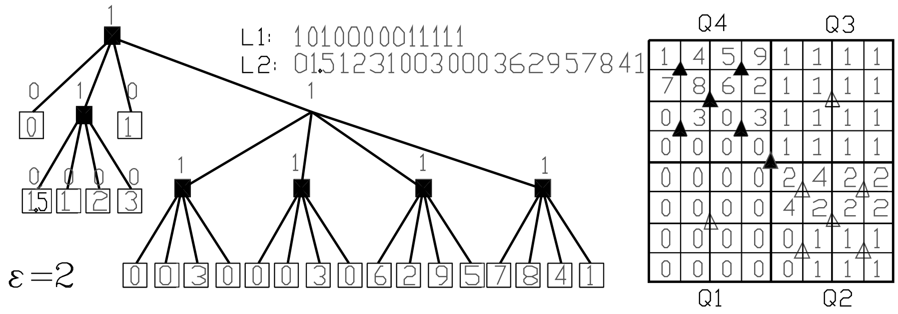

FIG. 1. *The image given on the right is represented by a quadtree (not balanced) on the left. The tolerance ε is set to 2. L_1 and L_2 are obtained by Algorithms 1 and 2. In L_1, the value 1 means an inner node and 0 indicates the leaf. Zeroes for the trivial quadrants of the size 1 are omitted (see Algorithm 1). The order of examining the quadrants is shown on the right (Q1,Q2,Q3,Q4). In Algorithm 3, both L_1 and L_2 are 2-dimensional boolean fields , where an information about the status of a quadtree region (leaf-blank triangle/inner node-full triangle) is stored in its centre.*

To illustrate this approach we introduce two simplified algorithms for creating and traversing the quadtree. The procedures needed to obtain a region are presented later.

Algorithm 1.

```
CONSTRUCT(root);
procedure CONSTRUCT(region)
begin
   if size_of_region=1
   {write(pixel_value) to L2;
    return;}
   if criterion is not fulfilled
     {write('1') to L1;
     for 4 children of the region
```

Algorithm 2.

```
l1=beginning of L1;
l2=beginning of L2;
TRAVERSE(root)
procedure TRAVERSE(region)
begin
if size_of_region=1
   {value=*l2; l2=l2+1;
    return};
if (*l1)=1
```

```
    CONSTRUCT(child)}
  else
   {write ('0') to L1;
    write(region_value) to L2;
   return};
end;
```

```
      for 4 children of the  region
       {l1=l1+1; TRAVERSE(child)};
else
  {value=*l2;
   l1=l1+1;l2=l2+1; return;}
end;
```

When information about neighbors is needed, this approach is known to be unsuitable. In our problem, this information is required during the creation the quadtree (which must be balanced i.e. for each region we must take into account the size of its neighbor) as well as during the traversion when the diffusion coefficient is calculated. To avoid these problems, we will modify the basic approach in two ways: to achieve a better correspondence between a quadtreee region in L_2 and its status in L_1, we will add to both lists some auxiliary positions, and, to maintain the balancedness, we change construction of L_1.

Let us take an image of an arbitrary size. To initialize L_2, we will embed it into a $2^N \times 2^N$ field (the additional positions are filled with some constant), for which we construct a quadtree encoded in the field L_1. The information, saying if some quadrant in L_2 is subdivided (1) or not (0), is stored in L_1 in such a way, that we are able to find it according to the position of the quadrant in L_2. Vice versa, according to the position of an item in L_1, we know, if it shows a status of some quadrant in L_2 and which. In the worst case, when no merging is possible, to encode the quadtree corresponding to the $2^N \times 2^N$ image, we need $\frac{4^N-1}{3}$ positions. The rest of the positions in L_1 is auxiliary. L_1 is the field of the size $(2^N+1)\times(2^N+1)$ and let us suppose, that it has been already constructed. During the quadtree traversion, it will be recursively subdivided according to the value in the centre of a $(2^k+1)\times(2^k+1)$ region (unlike L_2, quadrants of this field are overlapping). According to the position of a $(2^k+1)\times(2^k+1)$ quadrant in L_1, we can calculate the position of corresponding $2^k \times 2^k$ quadrant in L_2 and we can access the image value. Also the neighborhood information can be obtained. For a balanced quadtree, we need at most two tests in L_1 to find out the size and the position of the neighbor cell. If we embed the original image into $(2^N+1)\times(2^N+1)$ field instead of $2^N \times 2^N$ one, position of L_1 and L_2 quadrants directly corresponds - the positions of left lower corners of corresponding regions in L_1 and L_2 are the same. Such a situation is depicted in Figure 2 and used in Algorithm 3.

The last step is to set the indicator field L_1. We do not use the top-bottom approach of Algorithm 1, but the bottom-up way of merging the regions. New intensity value for merged cells is set to pixels' average. We start on the lowest (pixel) level of the structure with L_1 cleared and try to merge the 2×2 cells. In this approach, the lowest left corner of a 2×2 region in L_2 directly coresponds to the lowest left corner of a 3×3 stencil in L_1 (see Figure 2). If merging is not possible, the central position of the 3×3 stencil in indicator field is set to 1 to "mark" the tree node as inner and is sent to the corner positions of the stencil to maintain the balance condition. Then we continue on a higher level and try to merge 4×4 cells (e.g. quadruples of 2×2 merged regions), $(2^k+1)\times(2^k+1)$ etc.

To maintain the balance condition we use the fact, that corners on a lower level become middle points of sides of the L_1 stencils on a higher level. It enables us to control the merging also according to the size of neighboring cells: if some $(2^k+1)\times(2^k+1)$ indicator stencil on a higher level has "1" in some middle side position, the

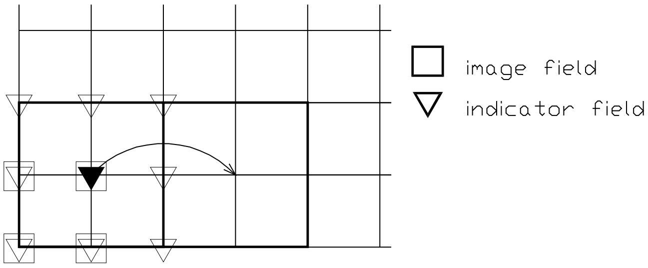

FIG. 2. *The mutual position of image values in L_2 and indicators in L_1 in a stencil on the lowest level. The picture shows the case, when both fields are of the same size.*

quadruple is not merged, even if its cells are within the tolerance ε. Otherwise the ratio of neighboring cells' size would be 1:4 or higher.

During the recursive process, the testing just the intensity difference of the cells in an inspected quadruple could cause cumulating of errors and in special cases resulting difference could be greater than ε. Such situation is depicted on the right of Figure 1 in quadrant Q_2. To avoid it, the merging criterion for testing a quadruple is changed. For every merged cell we remember, except for a new value, also minimum and maximum of all subcells and a modified *coarsening criterion* says: *we can merge four cells when the values in the corresponding center and middle positions in L_1 are zeroes, and difference of minimum and maximum for a given quadruple is bellow a prescribed threshold ε.* Working with an RGB image, we require the coarseing criterion for the inspected cells to be fulfilled in all three channels. In such a case we will work with one list L_1 and three lists L_2.

The following algorithm shows traversing of the quadtree provided that L_1 and L_2 have been already created and organized as described in the previous. The procedure *first_child* splits the inspected region and sets the new current region to the first child. The procedure *next_child* shifts to the following child: *next_child* from the last child returns NILL. If during the traversion the leaf element is reached, its diffusion coefficient is calculated by the procedure *coeff*.

Algorithm 3.

```
TRAVERSE(root);
procedure TRAVERSE(region)
begin
   if size_of_region=1
    {coeff(L2[position_of_region]);
     return;}
   if L1[center_of_region]=1
{for(child=first_child(region);child !=NIll;child=next_child(child)}
       TRAVERSE(child);return}
   else {coeff(L2[position_of_region]
          return};
 end;
```

It can be easily seen that having a $2^N \times 2^N$ image, the creation of the structure (i.e. setting L_1) needs $\frac{4^N-1}{3}$ comparisons of quadruples (or less, because if it is not possible to merge on a lower level, it is not possible on a higher level either). Any quadtree has $m = 1 + 3k$ leaf elements, for some $k \in N$, hanging in quadruples on k parent (inner) nodes. If c_1 is the cost of a procedure for getting the first child, c_2 is the cost of shifting to the next child and return to the parent element, then, having in mind the tree representation of the quadtree, it can be easily seen that the cost of traversing is about $\frac{m}{3}(c_1 + 4c_2)$. Let the quadtree is built upon an image given on a regular grid with M elements, and c_0 be the cost of the access to the next element in the regular grid. Then, for $r < \frac{(c_1+4c_2)}{3c_0}$, traversion of the quadtree is slower than going through the regular grid.

After creating the quadtree structure (by setting L_1), we calculate the diffusion coefficients by its recursive traversing. In such a way we create a system of linear equations, which is then solved using an iterative method with low memory requirements.

3. Finite volume scheme. Let τ_h be a mesh of Ω with cells p of measure $m(p)$ (we assume rectangular cells here). For every cell p we consider set of neighbours $N(p)$ consisting of all cells $q \in \tau_h$ for which common interface of p and q, denoted by e_{pq}, is of non-zero measure $m(e_{pq})$.

In the numerical scheme (3.7), we will provide computations in the series of discrete scale steps starting with $\overline{u}^0_{i_p}$, $p \in \tau_h$, corresponding to given intensities on the pixel structure of the initial discrete image. In the FVM, in every subsequent discrete scale step we get again a piecewise constant approximation $\overline{u}^n_{i_p}$, $p \in \tau_h$, $n = 1, 2, \ldots$ of the continuous solution. Convergence of such an approximations to a weak solution of (1.1)-(1.2) for a greyscale image, i.e. with i=1, provided the length of the discrete scale step and the size of the pixel tends to zero, is given in [7]. In [7], it is assumed that for every p, there exists a representative point $x_p \in p$, such that for every pair $p, q, q \in N(p)$, the vector $\frac{x_q - x_p}{|x_q - x_p|}$ is equal to unit vector n_{pq} which is normal to e_{pq} and oriented from p to q. (Let us note, that this assumption is not fulfilled for adaptive grids given by the coarsening algorithm). In [7] x_{pq} is the point of e_{pq} intersecting the segment $\overline{x_p x_q}$ and following coefficients are defined:

$$T_{pq} := \frac{m(e_{pq})}{|x_q - x_p|} \tag{3.3}$$

$$g^{\sigma,n}_{pq} := g(|\sum_{i=1}^{3} \nabla G_\sigma * \tilde{u}_i\,(x_{pq})\,|) \tag{3.4}$$

where $\tilde{u}_i$ is a periodic extension of a discrete color channel computed in n-th scale step.

To give the finite volume scheme for the adaptive grid, we modify the meaning of x_{pq} in (3.4) and definition (3.3) of T_{pq}. Let in the sequel x_{pq} be the middle point of the common boundary of two neighboring cells (with possibly non-equal measures). The definition of $g^{\sigma,n}_{pq}$ will then remain the same. In the definition of T_{pq} in (3.3), the value $|x_p - x_q|$ represents the distance used for an approximation of the normal derivative $\frac{u_q - u_p}{|x_q - x_p|}$. We can adjust this parameter in several ways. Here, we will introduce two ways of calculating T_{pq}.

Scheme 1. We can set $|x_p - x_q|$ to average size of two neighboring cells. Since our grids are balanced we put

$$T_{pq} := 1 \quad \text{if two inspected adjacent cells } p, q \text{ are of equal size,}$$

$$T_{pq} := \frac{2}{3} \quad \text{otherwise.} \tag{3.5}$$

Scheme 2. The second possibility which we consider is given by

$$T_{pq} = min\{l_p, l_q\} \tag{3.6}$$

where l_p and l_q are lengths of sides of two adjacent cells p, q (of possibly non-equal measure). It is like if we assume exchange of intensity between neighboring cells just in a strip of unit thickness along a boundary of a cell. This adjustment can be used for any grid, but in a case of a balanced grid T_{pq} is always equal to 1 or $\frac{1}{2}$.

The finite volume scheme on the adaptive grid for (1.1)-(1.2) is then written as follows:

Let $0 = t_0 \leq t_1 \leq ... \leq t_{N_{\max}} = T$ denote the scale discretization steps with $t_n = t_{n-1} + k$, where k is a discrete scale step. For $i = 1, 2, 3$ and $n = 0, ..., N_{\max} - 1$ we look for $\overline{u}_{i_p}^{n+1}$, $p \in \tau_h$, satisfying the system of linear equations

$$\left(\frac{m\,(p)}{k} + \sum_{q \in N(p)} g_{pq}^{\sigma,n} T_{pq} \right) \overline{u}_{i_p}^{n+1} - \sum_{q \in N(p)} g_{pq}^{\sigma,n} T_{pq} \overline{u}_{i_q}^{n+1} = \frac{m\,(p)}{k} \overline{u}_{i_p}^{n}. \tag{3.7}$$

This scheme is linear **semi-implicit** in scale, since a scale derivative is replace by backward difference and nonlinear terms of equation (1.1) are treated from the previous scale step while the linear terms are discretized on the current scale level. In every discrete scale step, the scheme gives linear systems which are symmetric and strictly diagonally dominant (with positive diagonal and negative numbers out of the diagonal) which guarantees existence of its unique solution and for which also L_∞ stability can be easy to prove. Moreover, it can be shown that for both schemes the mean value of the intensity is preserved.

In the case of a uniform square grid T_{pq} is equal to 1 and for a unit uniform grid with elements of unit size both schemes give the same result and represent the nonadaptive version of the algorithm. Nonadaptive versions for grids with cells of size bigger than 1 differ. If the coarsening process creates a uniform grid with cells of size l, the Scheme 2 works like Scheme 1, but with the scale step enlarged l times and thus diffusion is faster. Figure 3 demonstrates, that this is true also with adaptive grids with elements of arbitrary size.

In the scheme (3.7) we must compute the term (3.4). To compute the corresponding vector we can use the following property of the convolution

$$\frac{\partial (G_\sigma * \tilde{u})}{\partial x}\,(x_{pq}) = (\frac{\partial G_\sigma}{\partial x} * \tilde{u})\,(x_{pq})$$

and we get

$$\nabla G_\sigma * \tilde{u}_i\,(x_{pq}) = \sum_r \overline{u}_{i_r}^{n} \int_r \left(\frac{\partial G_\sigma}{\partial x}(x_{pq} - s)ds, \frac{\partial G_\sigma}{\partial y}(x_{pq} - s)ds \right). \tag{3.8}$$

Now the sum is restricted to the control volumes r inside $B_\sigma\,(x_{pq})$, the ball centered at x_{pq} with radius σ. The ball B_σ is given either by a support of the compactly supported smoothing kernel or it can represent a "numerical support" (a domain in which values of a function are above some treshold given e.g. by a computer precision) of

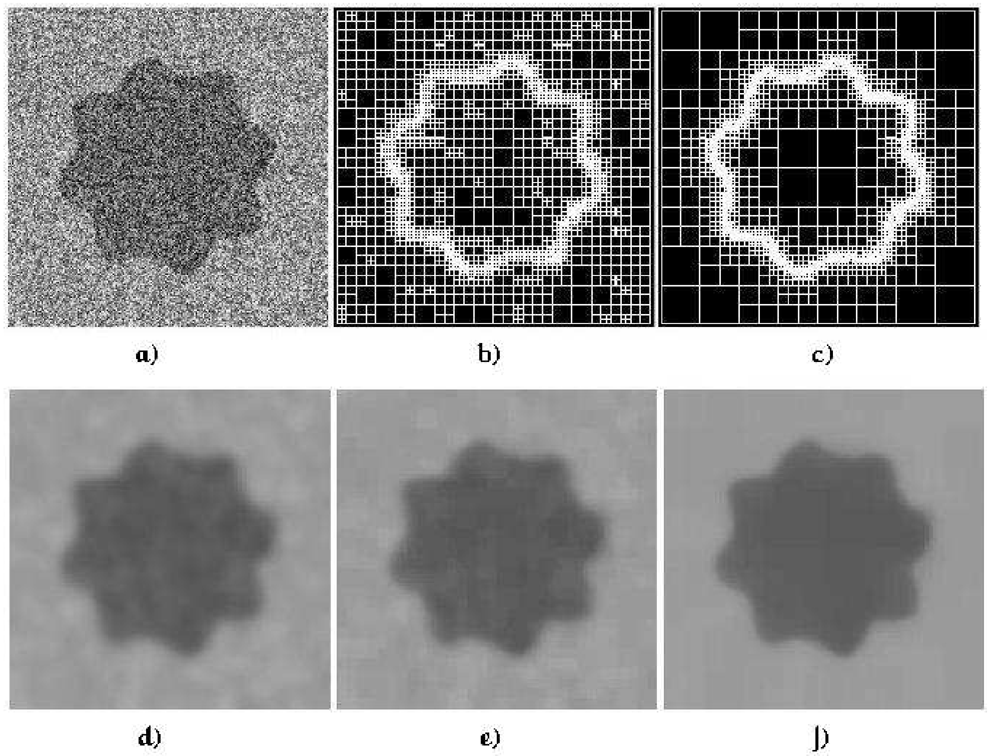

FIG. 3. *a) the original noisy image b) Scheme 1:grid after 20 steps c) Scheme 2: grid after 20 steps d) 20 steps of nonadaptive algorithm e) 20 steps of algorithm given by Scheme 1, f)20 steps of algorithm given by Scheme 2*

the Gauss function . In any case, just a finite sum in (3.8) is evaluated and coefficients of this sum, namely $\int_r \nabla G_\sigma (x_{pq} - s)\, ds$ can be precomputed in advance using a computer algebra system, e.g. Mathematica. Moreover, we can see that computing diffusion coefficients is signifficantly faster in the synchronized model, because they are computed only once. This is particularly desirable, when we work with σ covering several pixels, because it considerably reduces number of multiplication operations.

4. Numerical experiments. In this section we present experiments with some real as well as artificial images perturbed by various types of noise. In simulations, we use the function

$$g(s) = \frac{1}{1 + Ks^2}$$

with $K > 0$ and the convolution is realized with the kernel

$$G_\sigma(x) = \frac{1}{Z} e^{\frac{|x|^2}{|x|^2 - \sigma^2}},$$

where the constant Z is chosen so that G_σ has unit mass. In order to compute the diffusion coefficient $g_{pq}^{\sigma,n}$ we use the concept given in (3.8). In all numerical experiments we have chosen both the size of pixel and the scale step to be 1. Figures and graphs document results of multiscale analysis (iterative filtering) as well as adaptive computational grids. All experiments were done on PENTIUM II(400 MHz) with linux operating system.

Example 1. To every position of a double-valued image $\tilde{u}_i^0$ of the size 256×256 with intensity difference 150 we applied a noise by a transformation: if ψ is a random function generating values in $[0, 100]$, then for every pixel x and for $i = 1, 2, 3$

$$u_i^0(x) = MIN(255, MAX(0, \tilde{u}_i^0(x) - 50 + \psi).$$

The Figure 4 shows the original image perturbed by noise, the result of smoothing and the resulting mesh. The Figure 6 shows the decrease of number of unknowns and time needed for particular phases of the algorithm (building the quadtree by setting the indicator field, calculating the diffusion coefficients and solving the linear system) in time evolution for $\varepsilon = 0.03$, $\sigma = 0.5$, and $K = 10$. The initial image contains 65536 pixels. The total computational time for this example after 7 scale steps is 2.42 seconds for algorithm using Scheme 2. The Figure 5 shows the work of nonadaptive algorithm after 7, 10 and 12 scale steps with achieved times 6.22, 8.93 and 10.7 seconds. In this academic example we could choose ε considerably higher and thus achieve better time, e.g. for $\varepsilon = 0.3$ a comparable result is achieved after 3 time steps in 0.35 seconds. On the contrary, if we choose $\varepsilon = 0.004$, what corresponds to a difference of one pixel, the improvement is much smaller, because such is also the reduction of elements. The times achieved by the adaptive algorithm using Scheme 1 after 7, 10 and 12 scale steps were 5.7, 6.74 and 7.23 seconds , using Scheme 2 5.69, 6.78 and 8.495 seconds. Resulting images in both cases were similar to Figure 5.

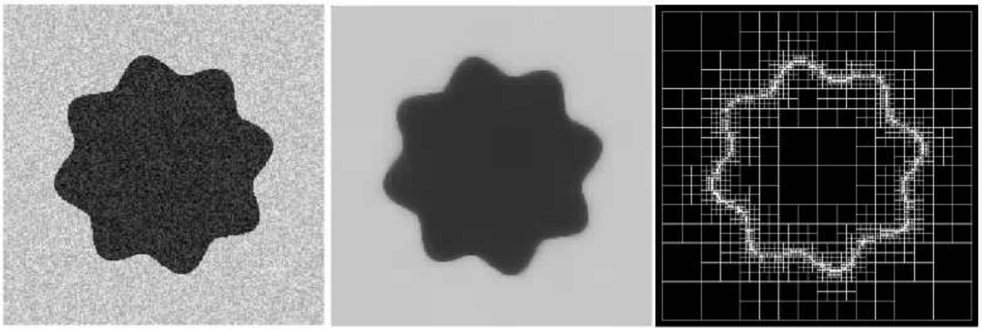

FIG. 4. *The initial noisy image, result of smoothing and an adaptive grid (Example 1)*

Example 2. The example 2 (Figure 7) works with the image size 512×512 with initial number of pixels 262144. The original double-valued image with intensity difference 150 was perturbed by 10% salt and pepper noise. For $\varepsilon = 0.025$, $\sigma = 0.5$, and $K = 10$ we need 10 scale steps of adaptive algorithm using Scheme 2 (Figure 6) with time 9.645 seconds , but 15 steps of nonadaptive algorithm (the times are 56.77 seconds for 15 scale steps and 39.065 for 10 scale steps).

Example 3. The experiment documented in Figure 8 was performed on the RGB image of the size 512×402 pixels. The picture is a result of scanning and has a significantly damaged blue channel (top of Figure 9). First, we show the original image and compare the work of synchronized and unsynchronized smoothing (Figure 8) performing 5 scale steps of nonadaptive algorithm with $K = 10$ and $\sigma = 1.5$. The example demonstrates better preserving of edges in synchronized model. The Figure 9 shows, that with help of red and green channels, which are of much better quality, the

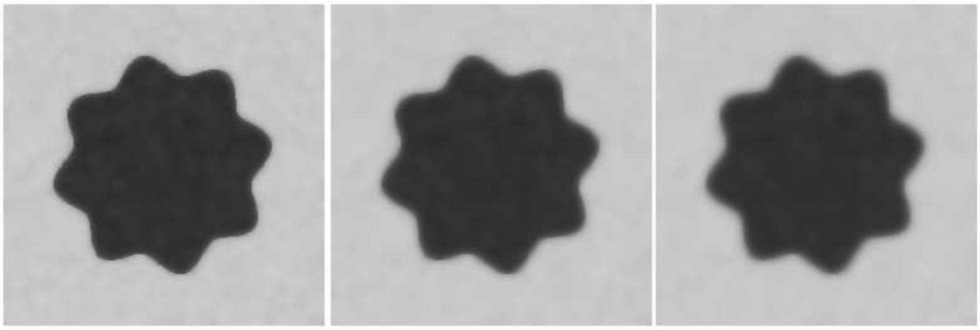

FIG. 5. *Result of smoothing by nonadaptive algorithm after 7, 10 and 12 scale steps.*

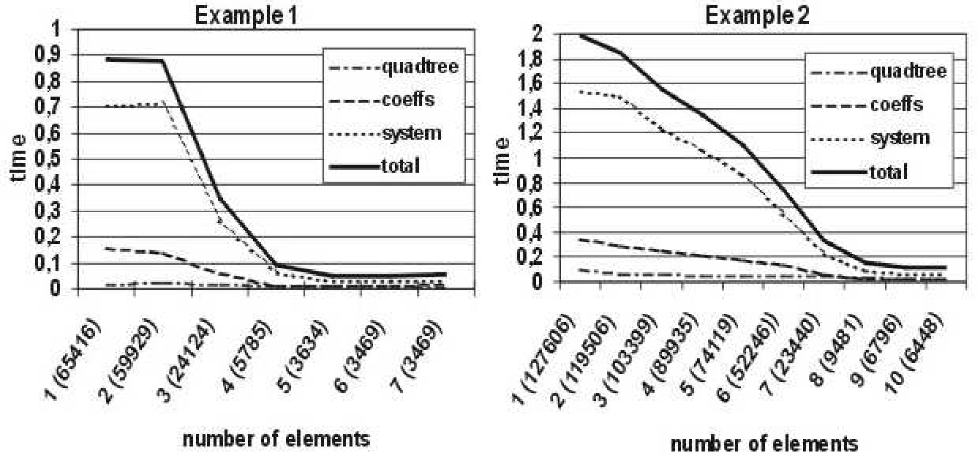

FIG. 6. *Times demonstrating work of adaptive algorithm using Scheme 2. In Example 1 the mean value for coeffs in nonadaptive version is 0.095s and for system 0.77s, in Example 2 coeffs took 0.37 s and system 1.7s.*

synchronized smoothing recovered the blue channel to the form shown in the bottom of the Figure, on the right.

Example 4. Channels of the RGB image in Figure 10 were independently disturbed by 10% salt and pepper noise. The size of the image is 433×512 pixels. For $\sigma = 0.5$ of the pixel's size, and $K = 10$ 28 steps of the adaptive algorithm required 280.48 seconds. The result is shown on the left in the bottom. For the adaptive version the image was completed to the size 512×512 with zero values. For this size, $K = 10, \sigma = 0.5, \varepsilon = 0.012$, and 26 scale steps we needed 108.75 seconds. In the last step, the initial number of unknowns 262144 was reduced to 34093. The Figure 10 demonstrates also a feature of Scheme 2 mentioned in Section 3 and demonstrated in the Figure 3: the diffusion for the adaptive algorithm given by Scheme 2 is faster than diffusion of nonadaptive algorithm.

REFERENCES

[1] L.ALVAREZ, F.GUICHARD, P.L.LIONS, J.M.MOREL, *Axioms and Fundamental Equations of Image Processing*, Arch. Rat. Mech. Anal. 123 (1993) pp. 200-257

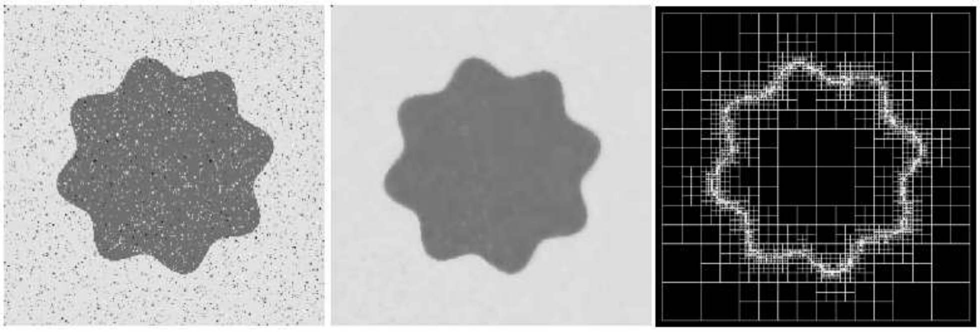

FIG. 7. *The initial image, result of smoothing and an adaptive grid (Example 2)*

[2] L.ALVAREZ, P.L.LIONS, J.M.MOREL, *Image selective smoothing and edge detection by nonlinear diffusion II*, SIAM J. Numer. Anal. 29 (1992) pp. 845-866

[3] L.ALVAREZ, J.M.MOREL, *Formalization and computational aspects of image analysis*, Acta Numerica (1994) pp. 1-59

[4] E.BÄNSCH, K.MIKULA, A coarsening finite element strategy in image selective smoothing, *Computing and Visualization in Science*, Vol.1, No.1 (1997) pp. 53-61

[5] F.CATTÉ, P.L.LIONS, J.M.MOREL, T.COLL, *Image selective smoothing and edge detection by nonlinear diffusion*, SIAM J.Numer.Anal. 29 (1992) pp. 182-193

[6] Z.KRIVÁ, K.MIKULA, *An adaptive finite volume scheme for solving nonlinear diffusion equations in image processing*, submitted

[7] K.MIKULA, N.RAMAROSY, *Semi-implicit finite volume scheme for solving nonlinear diffusion equations in image processing,* Numerische Mathematik, to appear

[8] M.OHLBERGER, M.RUMPF, *Hierarchical and adaptive visualization on nested grids*, Computing 59(4) (1997) pp. 269-285

[9] M.OHLBERGER, M.RUMPF, *Adaptive projection operators in multiresolutional scientific visualization*, IEEE Transactions on Visualization and Computer Graphics, 4(4) (1998)

[10] P.PERONA, J.MALIK, *Scale space and edge detection using anisotropic diffusion*, Proc. IEEE Computer Society Workshop on Computer Vision (1987)

[11] T. PREUSSER,M. RUMPF, *An Adaptive Finite Element Method for Large Scale Image Processing*, Proceedings of ScaleSpace'99 (1999) pp. 223-234

[12] J.WEICKERT, *Coherence-enhancing diffusion of colour images*, Image and Vision Computing 17 (1999) pp. 201-212

[13] R.WHITACKER, G.GERIG, *Vector-valued diffusion*, in B.M.t.M.Romemy(Ed): Geometry driven diffusion in computer vision, Kluwer(1994)

FIG. 8. *Example 3. The original image (in the top), result of 5 steps of the nonadaptive algorithm with synchronized smoothing (in the middle) and 5 steps of the nonadaptive algorithm with unsynchronized smoothing (in the bottom).*

FIG. 9. *In the top: red, green, and blue channels before smoothing (from left to the right). In the bottom: red, green, and blue channels (from left to the right)after 8 steps of synchronized smoothing by adaptive algorithm. In the middle: composition of smoothed channels.*

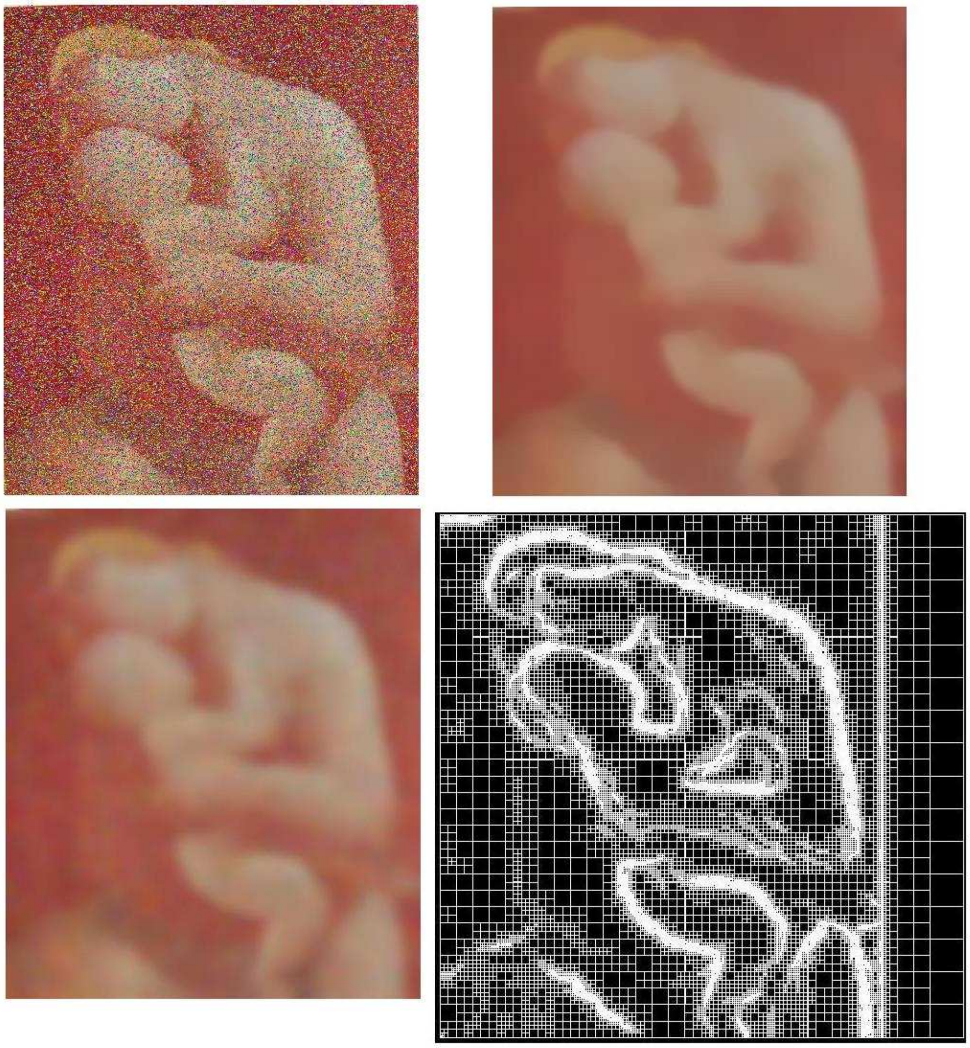

FIG. 10. *In the top from left to right: the original noisy image and image obtained by the adaptive smoothing. In the bottom from left to right: image obtained by the nonadaptive smoothing and a corresponding grid.*