PDE Based Image Diffusion and AOS

by Jarno Ralli

April 27, 2014

Abstract

This technical paper shows how AOS (Additive Operator Splitting) scheme, together with TDMA (Tri-Diagonal Matrix Algorithm), can be used for solving a non-linear, scalar valued diffusion equation. The paper shows how a continuous version of the diffusion equation is transferred into a 'discrete' version in order to be solved numerically. Much of the text is based on [3], and is presented here only for the sake of readability. Most part of the text deals with definitions, the idea here being that anyone familiar with diffusion should be able to understand the used notation and, thus, the formulations that follow. This text is supposed to be accompanied by a Matlab implementation of the code[1].

1 Introduction

Purpose of this document is to show how a non-linear scalar diffusion equation can be solved efficiently using a block-solver called Tri-Diagonal Matrix Algorithm (TDMA). Equation (1) shows the diffusion equation that we are going to use in this paper. The idea of this equation is to diffuse image information in homogeneous areas of the image. Homogeneous here means areas where the pixel values are roughly the same. Therefore, we blend image information *inside* objects while borders (also known as edges) are left intact. As an example, suppose we have two different objects in the image, a dog and a cat. We want to diffuse the image inside the dog- and the cat objects but we do not want to diffuse at the borders separating the dog from the cat. As will be shown later on, diffusion is controlled by using a function $g(x, y, t)$ that controls the diffusion weights for each pixel.

$$\frac{\partial I_k}{\partial t} = DIV\Big(g(x, y, t)\nabla I_k\Big) \tag{1}$$

where k refers to the channel in question, $g(x, y, t)$ defines the (scalar) diffusion weights, and $\nabla := [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}]$ is the spatial gradient. Since $g(x, y, t)$ is a function of t, the diffusion is non-linear. What this means is that the diffusion weights defined by the function $g(x, y, t)$ change with respect to time t. One possible $g(x, y, t)$ is, for example:

$$g(x, y, t) = \frac{1}{1 + \left(\frac{||\nabla I(x,y,t)||}{K}\right)^2} \tag{2}$$

[1] `http://www.jarnoralli.fi/joomla/code/diffusion-code`

where $||\nabla I(x,y,t)||$ refers to magnitude of the image derivatives (obtains greater values when there is a change in the image). K is a coefficient that is used for controlling the diffusion weight magnitudes with respect to the image derivatives. Function (2) behaves as shown in (3):

$$g(x,y,t) \approx \begin{cases} 0 & \text{, when } ||\nabla I|| \to \inf \\ 1 & \text{, when } ||\nabla I|| \to 0 \end{cases} \tag{3}$$

From (3) we can see that function obtaines values near zero where we have strong image derivatives. On the other hand, the function obtains values near 1 in homogeneous areas of the image. Fig. 1 shows how the diffusion results look like in practice.

(a) before (b) after

Figure 1: (a) displays the original image before applying diffusion while (b) shows the image after diffusion has been applied. As it can be observed from the images, since diffusion is almost halted at image edges these maintain crisp while insides of the objects have been diffused.

Principla idea of this document is to show the reader how the diffusion equation can be solved efficiently using an efficient block-solver called TDMA (Tri-Diagonal Matrix Algorithm). However, before going any further the mathematical notation is introduced in Chapter 2. Since the notation can become somewhat awkward to read, I think it is a good idea to first introduce the notation used in the rest of the text. After this, in Chapter 3 I show how the diffusion equation can be discretized so that it can be solved by using numerical methods. In Chapter 5 we derive the AOS (Additive Operator Splitting) scheme which allows us to use the TDMA-solver for solving the equation. This chapter also contains an example of a Matlab-code.

2 Used Notation

We consider the image to be a continuous function with $I(x,y,k) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^{K+}$, where the domain of the image is $\Omega = \mathbb{R} \times \mathbb{R}$ and K defines the number of channels (e.g. $K = 3$ in the case of RGB images). It is in this regular domain where our PDEs are defined. However, in order for us to solve the PDEs, they need to be discretised first. Also, the kind of images that we are dealing with are, in fact, discretised versions that we receive from the imaging devices, such as digital- or thermal cameras. We define a discretisation grid as given in (4):

$$G_h := \{(x, y) \mid x = x_i = ih_x,\, y = y_j = jh_y\,;\, i, j \in \mathbb{Z}\} \tag{4}$$

where $h = (h_x, h_y)$ is a discretisation parameter. With the discretisation grid, the domain of the discretised images can be defined as $\Omega_h = \Omega \cap G_h$. Instead of $I(x, y) = I(ih_x, jh_y)$, we typically use $I_{i,j}$ when pointing to the pixels.

Sometimes positions on a grid are given using both cardinal- and inter-cardinal directions, as defined in Figure 2. The idea is to simplify the notation of the discretised versions of the equations which can be rather messy.

NW	N	NE
W	C	E
SW	S	SE

Figure 2: Directions on a grid. To simplify the notation, both cardinal- and inter-cardinal directions are used. Here W, N, E, S, and C refer to west, north, east, south, and centre, respectively.

2.1 Pixel Numbering Schemes

As it was already mentioned, we can refer to any pixel in the image by using $I_{i,j}$, where $0 \leq i \leq m$, $0 \leq j \leq n$. Here the discretisation parameter $h = (h_x, h_y)$ has been chosen so that the discretised domain is $\Omega_h : [1, m] \times [1, n]$. Another particularly useful way is to think of the discretised image as a vector $I \in \mathbb{R}^N$. Now, the components of the vector are $I_{\mathcal{I}}$ where $\mathcal{I} \in \{1, \ldots, N\}$ and N is the number of pixels in image. This second numbering scheme is useful for algorithmic descriptions, and in matrix notation, as will be shown later on.

Figure 3 depicts both column- and row-wise traversing order of pixels. Depending on the solver, either column- or row-wise traversing, or a combination of both (for example, first column- and then row-wise), can be used. An interested reader is pointed to [4] for more information.

1	4	7
2	5	8
3	6	9

(a) Column wise.

1	2	3
4	5	6
7	8	9

(b) Row wise.

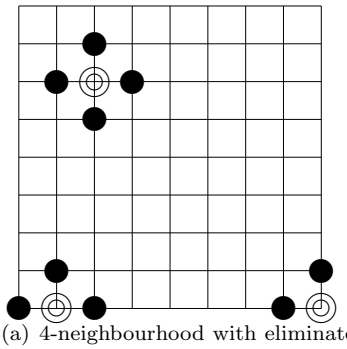
Figure 3: Column- and row-wise pixel orderings. Here, $\mathcal{I} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

.

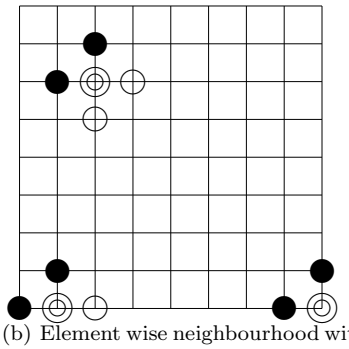
2.2 Pixel Neighbourhoods

In order to simplify the notation, for example in algorithmic descriptions, we define different kinds of pixel neighbourhoods. With pixel neighbourhood we mean image pixels $I_{\mathcal{J}}$ that are neighbours of a pixel of interest $I_{\mathcal{I}}$. The neighbourhoods are slightly different depending on if we are talking about a element-

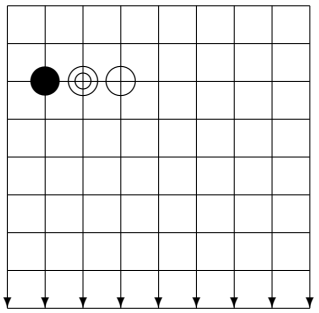
or a block-wise solver. By element wise solver we mean a Jacobi or a Gauss-Seidel type iterative solvers, that search for the solution for a single element at a time. On the other hand, block type solvers search for a solution for a group of elements (or a block). However, since the neighbourhoods have the same function in both the above mentioned cases, we use the same neighbourhood operator to denote the neighbours. It should be clear from the structure of the solver which kind of a neighbourhood is in question. $\mathcal{J} \in N(\mathcal{I})$ denotes the set of neighbours $\mathcal{J}$ of $\mathcal{I}$, as seen in Figure 4 (a).



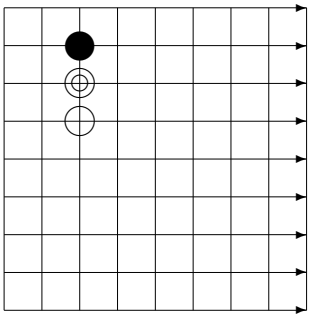
(a) 4-neighbourhood with eliminated boundary conditions.



(b) Element wise neighbourhood with eliminated boundary conditions.



(c) Block wise, column ordering.



(d) Block wise, row ordering.

Figure 4: Pixel neighbourhoods, where central pixel, $\mathcal{I}$, is denoted with a double circle. (a) Painted circles denote neighbouring pixels $\mathcal{J}$ that belong to the neighbourhood defined by $\mathcal{J} \in N(\mathcal{I})$. (b) Painted circles denote pixels $\mathcal{J}$ that belong to the neighbourhood defined by $\mathcal{J} \in N^-(\mathcal{I})$, while unpainted circles denote pixels $\mathcal{J}$ that belong to the neighbourhood defined by $\mathcal{J} \in N^+(\mathcal{I})$. (c)-(d) Painted circles denote pixels $\mathcal{J}$ that belong to the neighbourhood defined by $\mathcal{J} \in N^-(\mathcal{I})$, while unpainted circles denote pixels $\mathcal{J}$ that belong to the neighbourhood defined by $\mathcal{J} \in N^+(\mathcal{I})$. Processing order is defined by the arrows. Due to the eliminated boundary conditions 'scheme', the pixel neighbourhood operators only point to valid neighbours, as shown in (a) and (b).

Pixel wise. $\mathcal{J} \in N^-(\mathcal{I})$ denotes the set of neighbours $\mathcal{J}$ of $\mathcal{I}$ with $\mathcal{J} < \mathcal{I}$ (painted circles in Figure 4 (b)), and $\mathcal{J} \in N^+(\mathcal{I})$ denotes the neighbours $\mathcal{J}$ of $\mathcal{I}$ with $\mathcal{J} > \mathcal{I}$ (unpainted circles in Figure 4 (b)).

Block wise. $\mathcal{J} \in N^-(\mathcal{I})$ denotes the neighbours $\mathcal{J}$ of $\mathcal{I}$ with $\mathcal{J} < \mathcal{I}$ (painted circles in Figure 4 (c)-(d)). Since we run the block wise solver both column- and row-wise, depending on the direction, the neighbour(s) defined by this neighbourhood operator changes. $\mathcal{J} \in N^+(\mathcal{I})$ denotes the neighbours $\mathcal{J}$ of $\mathcal{I}$ with $\mathcal{J} > \mathcal{I}$ (painted circles in Figure 4 (c)-(d)). Again, the actual neighbours defined by this operator depends on the direction of the solver.

3 Scalar Valued Diffusion

The basic formula describing a scalar valued diffusion is:

$$\frac{\partial I_k}{\partial t} = DIV\Big(g(x,y,t)\nabla I_k\Big) \tag{5}$$

where k refers to the channel in question, $g(x,y,t)$ defines the (scalar) diffusion weights, and $\nabla := [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}]$ is the spatial gradient. Since $g(x,y,t)$ is a function of t, the diffusion is non-linear.

3.1 Implicit Scheme

For discretisation, we use Euler backward, semi-implicit method, and obtain the following:

$$\frac{(I_k)^{t+1} - I_k^t}{\tau} = DIV\Big(g^t\nabla(I_k)^{t+1}\Big) \tag{6}$$

where $g^t = g(x,y,t)$. In other words, we use values of g, available at time t, when resolving for a new value at time $t+1$.

4 Finite Difference Discretisation

4.1 Finite Difference Operators

Before going any further, we remind the reader of the 'standard' finite difference operators, which are used for approximating derivatives. Here, the function of interest, for which we want to find derivatives, is defined as $f(x,y) : \Omega \to \mathbb{R}$.

1. First order forward difference is given by:

$$D_x^+ f(x,y) = f_x^+(x,y) = \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x} \tag{7}$$

2. First order backward difference is given by:

$$D_x^- f(x,y) = f_x^-(x,y) = \frac{f(x,y) - f(x-\Delta x, y)}{\Delta x} \tag{8}$$

3. First order central difference is given by:

$$D_x^0 f(x,y) = f_x^0(x,y) = \frac{f(x+\frac{1}{2}\Delta x, y) - f(x-\frac{1}{2}\Delta x, y)}{\Delta x} \tag{9}$$

4. Second order central difference is given by:

$$DD_x^0 f(x,y) = f_{xx}^0(x,y) = \frac{f(x+\Delta x, y) - 2f(x,y) + f(x-\Delta x, y)}{\Delta x^2} \quad (10)$$

where in f_x the sub-index (here x) indicates with respect to which variable the function has been differentiated. The above formulations can be simplified further if we assume a uniform grid with $\Delta x = \Delta y = 1$. The difference operators shown here approximate derivatives with respect to x. By switching the roles of x and y, corresponding difference operators for approximating derivatives with respect to y can be obtained easily. In the case of images containing several channels (e.g. RGB-images), derivatives are approximated separately for each channel.

4.2 Discretisation of DIV Operator

Now that we know how to approximate first and second order derivatives, we can discretise the divergence, DIV, operator. Conceptually, we have two different cases:

$$\begin{aligned} &DIV\Big(\nabla f\Big) \\ &DIV\Big(g(x,y,t)\nabla f\Big) \end{aligned} \quad (11)$$

Here, physical interpretation of the divergence is, in a sense, that of *diffusion* [2]. In the case of $DIV\,(\nabla f)$, diffusivity is the same in each direction, whereas in the case of $DIV\,(g(x,y,t)\,\nabla f)$, diffusivity is defined (or controlled) by the function g and is not necessarily the same in all the directions. Mathematically, for a differentiable vector function $F = U\vec{i} + V\vec{j}$, divergence operator is defined as:

$$DIV\Big(F\Big) = \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} \quad (12)$$

In other words, divergence is a sum of partial derivatives of a differentiable vector function. Therefore, in our case, we have the following.

$$\begin{aligned} DIV\Big(\nabla f\Big) &= \frac{\partial}{\partial x}\big(f_x\big) + \frac{\partial}{\partial y}\big(f_y\big) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \Delta f \\ DIV\Big(g(x,y,t)\nabla f\Big) &= \frac{\partial}{\partial x}\big(g(x,y,t)f_x\big) + \frac{\partial}{\partial y}\big(g(x,y,t)f_y\big) = \nabla g \,.\, \nabla f + g\Delta f \end{aligned} \quad (13)$$

Now, by simply using the finite differences introduced above, one way of discretising the divergence terms in (13) is using the central difference. First we apply the central difference and then the forward- and the backward differences for approximating the corresponding derivatives. The 'trick' here is to realise that $(f_x)(x+0.5, y)$ is actually the forward difference $D_x^+ f(x)$, while $(f_x)(x-0.5, y)$ is the backward difference $D_x^- f(x)$. Equations (14), and (15) show the

discretisations for $DIV(\nabla f)$, and $DIV(g(x,y,t)\nabla f)$, respectively. This is the same discretisation as in the famous paper by Perona and Malik [2].

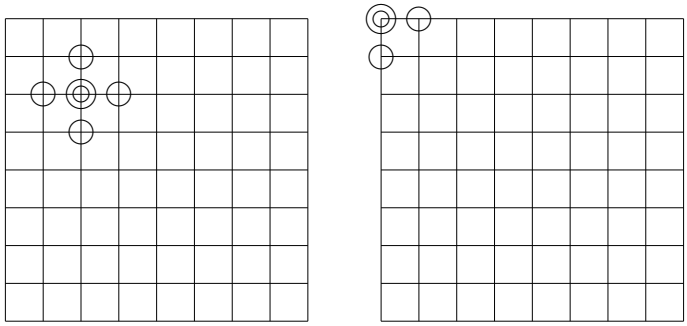
$$\begin{aligned}\frac{\partial}{\partial x}\Big(f_x\Big)(x,y)+\frac{\partial}{\partial y}\Big(f_y\Big)(x,y) =&\Big(f_x\Big)(x+0.5,y)-\Big(f_x\Big)(x-0.5,y)\\ &+\Big(f_y\Big)(x,y+0.5)-\Big(f_y\Big)(x,y-0.5)\\ =&f(x+1,y)-f(x,y)+f(x-1,y)-f(x,y)\\ &+f(x,y+1)-f(x,y)+f(x,y-1)-f(x,y)\\ =&\nabla_E f+\nabla_W f+\nabla_S f+\nabla_N f\end{aligned} \tag{14}$$

where $\nabla_{\{W,N,E,S\}}f$ denotes the difference in the directions given by W,N,E,S. As it was already mentioned, first we apply first order central difference on $f_x(x,y)$, and thus obtain $D_x^0 f_x(x,y)=\Big(f_x\Big)(x+0.5,y)-\Big(f_y\Big)(x-0.5,y)$.

$$\begin{aligned}\frac{\partial}{\partial x}\Big(gf_x\Big)(x,y)+\frac{\partial}{\partial y}\Big(gf_y\Big)(x,y) =&\Big(gf_x\Big)(x+0.5,y)-\Big(gf_x\Big)(x-0.5,y)\\ &+\Big(gf_y\Big)(x,y+0.5)-\Big(gf_y\Big)(x,y-0.5)\\ =&g(x+0.5,y)\Big(f(x+1,y)-f(x,y)\Big)\\ &+g(x-0.5,y)\Big(f(x-1,y)-f(x,y)\Big)\\ &+g(x,y+0.5)\Big(f(x,y+1)-f(x,y)\Big)\\ &+g(x,y-0.5)\Big(f(x,y-1)-f(x,y)\Big)\\ =&g_E\nabla_E f+g_W\nabla_W f+g_S\nabla_S f+g_N\nabla_N f\end{aligned} \tag{15}$$

where $g_{\{W,N,E,S\}}$ denotes diffusivity in the directions given by W,N,E,S. As it can be observed from Equation (15), we need to approximate the diffusivity between the pixels. A simple '2-point' approximation would be the average between neighbouring pixels, for example $g(x+0.5,y)=[g(x+1,y)+g(x,y)]/2$. A more precise approximation, leading to better results, is a '6-point' approximation of Brox [1].

As it was already mentioned above, physical interpretation of the divergence is, in a sense, *diffusion*: the divergence operator introduces a 'connectivity' between the pixels. This simply means, as will be shown later on, that a solution at any position (i,j) will depend on the solution at neighbouring positions. Because of this kind of a dependency of the solution between the adjacent positions, variational correspondence methods are said to be 'global'. This kind of connectivity is problematic at image borders, where we do not have neighbours anymore. In order to deal with this problem, we use a scheme called *eliminated boundary conditions*, shown in Figure 5.



(a) (b)

Figure 5: Double circle denotes the position of interest while single circles are the neighbouring positions W, N, E, S; (b) shows the eliminated boundary conditions.

4.3 Discretised Diffusion Equation

In order to solve Equation 6, we need to discretise the divergence operator. We start by marking the positions of the pixels of interest with (i, j):

$$\frac{(I_k)_{i,j}^{t+1} - (I_k)_{i,j}^{t}}{\tau} = DIV\left(g^t \nabla I_k^{t+1}\right) \tag{16}$$

After this we discretise the DIV operator, as given by Equation 15, and obtain the following:

$$\begin{aligned}(I_k)_{i,j}^{t+1} - (I_k)_{i,j}^{t} =& \tau g_N^t \left((I_k)_{i-1,j}^{t+1} - (I_k)_{i,j}^{t+1}\right) \\ &+ \tau g_S^t \left((I_k)_{i+1,j}^{t+1} - (I_k)_{i,j}^{t+1}\right) \\ &+ \tau g_W^t \left((I_k)_{i,j-1}^{t+1} - (I_k)_{i,j}^{t+1}\right) \\ &+ \tau g_E^t \left((I_k)_{i,j+1}^{t+1} - (I_k)_{i,j}^{t+1}\right)\end{aligned} \tag{17}$$

The only thing left to do, is arrange the terms:

$$\begin{aligned}(I_k)_{i,j}^{t+1}\left(1 + \tau(g_N^t + g_S^t + g_W^t + g_E^t)\right) =& (I_k)_{i,j}^{t} \\ &+ \tau g_N^t \left((I_k)_{i-1,j}^{t+1}\right) \\ &+ \tau g_S^t \left((I_k)_{i+1,j}^{t+1}\right) \\ &+ \tau g_W^t \left((I_k)_{i,j-1}^{t+1}\right) \\ &+ \tau g_E^t \left((I_k)_{i,j+1}^{t+1}\right)\end{aligned} \tag{18}$$

4.4 Matrix Format

While Equation (18) shows equation to be solved for a pixel position (i, j), we can write the system equations in matrix/vector format, covering the whole

image, as given in (20). Now, the components of the vector are $I_{\mathcal{I}}$ where $\mathcal{I} \in \{1, \ldots, N\}$ and N is the number of pixels in image. We use $\mathcal{I}, \mathcal{J}$ also to mark the positions in the system matrix A given in (20). This is done in order to convey clearly the idea that the domains of the discretised images and the system matrices are different. If the domain of the discretised image is $\Omega_h : [1, m] \times [1, n]$ (discrete image with m columns and n rows), the system matrix A is defined on $[m \cdot n] \times [m \cdot n]$ (here $\cdot$ denotes multiplication). Now, we can write the Euler forward, semi-implicit formulation in a vector/matrix format as follows:

$$\frac{(\mathbf{I}_k)^{t+1} - (\mathbf{I}_k)^t}{\tau} = A\Big((\mathbf{I}_k)^t\Big)\mathbf{I}_k^{t+1} \tag{19}$$

where $\mathbf{I}_k := (I_k)_{\mathcal{I}}$ with $\mathcal{I} = [1 \ldots N]$, and k refers to the channel in question (e.g. R, G or B). The system matrix $A\Big((\mathbf{I}_k)^t\Big)$ is defined as follows:

$$\begin{aligned} &A\Big((\mathbf{I}_k)^t\Big) = [a^t_{\mathcal{I},\mathcal{J}}] \\ &a^t_{\mathcal{I},\mathcal{J}} := \begin{cases} -\tau g^t_{\mathcal{J}\sim\mathcal{I}} & [\mathcal{J} \in N(\mathcal{I})], \\ 1 + \sum\limits_{\substack{\mathcal{J}\in N^-(\mathcal{I}) \\ \mathcal{J}\in N^+(\mathcal{I})}} \tau g^t_{\mathcal{J}\sim\mathcal{I}} & (\mathcal{J} = \mathcal{I}), \\ 0 & (\text{otherwise}) \end{cases} \end{aligned} \tag{20}$$

where $g^t_{\mathcal{J}\sim\mathcal{I}}$ refers to the 'diffusion' weight between pixels $\mathcal{J}$ and $\mathcal{I}$ at time t. In other words, these refer to the $g_{\{W,\,N,\,E,\,S\}}$ seen previously. Equation (21) gives an example of how the system matrix A would look like for a 3×3 size image. Here C and N are block matrices that refer to the 'central' and the 'neighbouring' matrices, correspondingly.

$$\begin{aligned} A &= \begin{bmatrix} C & N & 0 \\ N & C & N \\ 0 & N & C \end{bmatrix} \\ C &= \begin{bmatrix} 1 + \sum\limits_{\substack{\mathcal{J}\in N^-(\mathcal{I}) \\ \mathcal{J}\in N^+(\mathcal{I})}} \tau g^t_{\mathcal{J}\sim\mathcal{I}} & -\tau g^t_{\mathcal{J}\sim\mathcal{I}} & 0 \\ -\tau g^t_{\mathcal{J}\sim\mathcal{I}} & 1 + \sum\limits_{\substack{\mathcal{J}\in N^-(\mathcal{I}) \\ \mathcal{J}\in N^+(\mathcal{I})}} \tau g^t_{\mathcal{J}\sim\mathcal{I}} & -\tau g^t_{\mathcal{J}\sim\mathcal{I}} \\ 0 & -\tau g^t_{\mathcal{J}\sim\mathcal{I}} & 1 + \sum\limits_{\substack{\mathcal{J}\in N^-(\mathcal{I}) \\ \mathcal{J}\in N^+(\mathcal{I})}} \tau g^t_{\mathcal{J}\sim\mathcal{I}} \end{bmatrix} \\ N &= \begin{bmatrix} -\tau g^t_{\mathcal{J}\sim\mathcal{I}} & 0 & 0 \\ 0 & -\tau g^t_{\mathcal{J}\sim\mathcal{I}} & 0 \\ 0 & 0 & -\tau g^t_{\mathcal{J}\sim\mathcal{I}} \end{bmatrix} \end{aligned} \tag{21}$$

From (20) we can see how the matrix A looks like for a 3×3 size image: it is a block tridiagonal square matrix, of size 9×9, that has non-zero components only on the main diagonal and on the diagonals adjacent to this. Therefore, unless the image is very small, it is infeasible to solve the system by inverting A directly. Instead, we search for a solution using iterative methods.

In the following we can see how A would look like using different 'notation'. It is interesting to see how changing the traversing order changes the neighbours on the diagonals next to the main diagonal. Sub matrices with grey background refer to the sub matrix C given in Equation 21

C	S	0	E	0	0	0	0	0
N	C	S	0	E	0	0	0	0
0	N	C	0	0	E	0	0	0
W	0	0	C	S	0	E	0	0
0	W	0	N	C	S	0	E	0
0	0	W	0	X	C	0	0	E
0	0	0	W	0	0	C	S	0
0	0	0	0	W	0	N	C	S
0	0	0	0	0	W	0	N	C

(a) Column-wise traversing.

C	E	0	S	0	0	0	0	0
W	C	E	0	S	0	0	0	0
0	W	C	0	0	S	0	0	0
N	0	0	C	E	0	S	0	0
0	N	0	W	C	E	0	S	0
0	0	N	0	X	C	0	0	S
0	0	0	N	0	0	C	E	0
0	0	0	0	N	0	W	C	E
0	0	0	0	0	N	0	W	C

(b) Row-wise traversing.

Figure 6: Table like representation of the system matrix A for both column- and row-wise ordering. Here W, N, E, S and C refer to the neighbours given by the cardinal directions (West, North, East and South), while C refers to the 'Centre'.

5 AOS

With the vector/matrix format in place, we can now formulate the 'additive operator splitting' scheme by Weickert et al. [5]. In order to simplify the notation, we write A instead of $A\Big((\mathbf{I}_k)^t\Big)$. Id refers to the identity matrix. Therefore, we have:

$$(\mathbf{I}_k)^{t+1} = (\mathbf{I}_k)^t + \tau A \mathbf{I}_k^{t+1} \tag{22}$$

From which $(\mathbf{I}_k)^{t+1}$ can be solved as follows:

$$(\mathbf{I}_k)^{t+1} = \Big(Id - \tau A\Big)^{-1} \mathbf{I}_k^t \tag{23}$$

Now, we 'decompose' A so that $A = \sum_{l=1}^{m} A_l$, which allows as to write the above equation as:

$$(\mathbf{I}_k)^{t+1} = \left(\sum_{l=1}^{m} \frac{1}{m} Id - \tau \sum_{l=1}^{m} A_l \right)^{-1} \mathbf{I}_k^t \tag{24}$$

where m is the number of dimensions (in our case $m = 2$). Previous equation can be written, using only a single summation operator, as:

$$(\mathbf{I}_k)^{t+1} = \left(\sum_{l=1}^{m} \frac{1}{m} \Big(Id - \tau m A_l \Big) \right)^{-1} \mathbf{I}_k^t \tag{25}$$

The idea of writing A as $A = \sum_{l=1}^{m} A_l$ is based on the traversing order of A as shown in Fig. 6. When constructing each A_l (here l can be though of referring to the traversing order based on the dimension), we only take into account the neighbours on the diagonals next to the main diagonal and discard the rest. Physically this can be interpreted as diffusing separately along the vertical and horizontal directions. When constructing the matrices A_l, one must be careful with the 'central' elements, Equation (21), so that only the neighbours on the diagonals next to the main diagonal are taken into account. This is apparent from Equation (33).

Equation (25) has interesting 'form' in the sense that the 'system matrix' is decomposed. The problem is that the decomposed system matrix is inside the $()^{-1}$ operator. Instead, we would like to construct the solution in parts as follows:

$$(\mathbf{I}_k)^{t+1} = \sum_{l=1}^{m} \left(\frac{1}{m} \Big(Id - \tau m A_l \Big) \right)^{-1} \mathbf{I}_k^t \tag{26}$$

The problem is that the right hand sides of Equations (25) and (26) are not equal, as can be easily verified. Therefore, we pose the question if there exists a simple variable x, when used to multiply the right hand side of (26), would make these equal:

$$\left(\sum_{l=1}^{m} \frac{1}{m} \underbrace{\Big(Id - \tau m A_l \Big)}_{B} \right)^{-1} \mathbf{I}_k^t = x \sum_{l=1}^{m} \left(\frac{1}{m} \underbrace{\Big(Id - \tau m A_l \Big)}_{B} \right)^{-1} \mathbf{I}_k^t \tag{27}$$

The above can be simplified into:

$$B^{-1} = x m^2 B^{-1} \tag{28}$$

And, thus we have:

$$x = \frac{1}{m^2} \tag{29}$$

Based on this, in order to use the 'additive operator splitting' scheme given by Equation (26), we multiply the right hand side with $\frac{1}{m^2}$, and obtain the following:

$$(\mathbf{I}_k)^{t+1} = \frac{1}{m^2}\sum_{l=1}^{m}\left(\frac{1}{m}\Big(Id - \tau m A_l\Big)\right)^{-1}\mathbf{I}_k^t \tag{30}$$

which is the same as:

$$(\mathbf{I}_k)^{t+1} = \sum_{l=1}^{m}\left(mId - \tau m^2 A_l\right)^{-1}\mathbf{I}_k^t \tag{31}$$

As an example, if $l = 2$ (2D), then we would have:

$$(\mathbf{I}_k)^{t+1} = \Big(2Id - 4\tau A_1\Big)^{-1}\mathbf{I}_k^t + \Big(2Id - 4\tau A_2\Big)^{-1}\mathbf{I}_k^t \tag{32}$$

Now, as it can be understood, the whole idea of this scheme is to bring the equations to a 'simpler' form, allowing us to use efficient block-wise solvers, such as TDMA (TriDiagonal Matrix Algorithm).

5.1 TDMA

TDMA. Tridiagonal matrix algorithm (TDMA) is a simplified form of Gaussian elimination, that can be used for solving tridiagonal systems of equations. In matrix/vector format this kind of a system can be written as in (33)

$$\underbrace{\begin{bmatrix} b_1 & c_1 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 \\ 0 & a_3 & b_3 & \ddots & 0 \\ 0 & 0 & \ddots & \ddots & c_{N-1} \\ 0 & 0 & 0 & a_N & b_N \end{bmatrix}}_{A}\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix}}_{d} \tag{33}$$

The algorithm consists of two steps: the first (forward) sweep eliminates the a_i, while the second (backward) sweep calculates the solution. Equation (34) introduces the forward sweep, while Equation (35) shows the backward sweep.

$$\begin{aligned} c'_i &= \begin{cases} \dfrac{c_1}{b_1} & , i = 1 \\ \dfrac{c_i}{b_i - c'_{i-1}a_i} & , i = 2, 3, ..., N-1 \end{cases} \\ d'_i &= \begin{cases} \dfrac{d_1}{b_1} & , i = 1 \\ \dfrac{d_i - d'_{i-1}a_i}{b_i - c'_{i-1}a_i} & , i = 2, 3, ..., N \end{cases} \end{aligned} \tag{34}$$

$$\begin{aligned} x_N &= d'_N \\ x_i &= d'_i - c'_i x_{i+1} \quad , i = N-1, N-2, ..., 1 \end{aligned} \tag{35}$$

Physical interpretation of the terms a_i and b_i is that they are diffusion weights, i.e. how much the neighbouring solutions are taken into account.

Figures 7 and 8 show both the horizontal- and vertical traversing orders, and how the coefficients a_i and c_i are chosen.

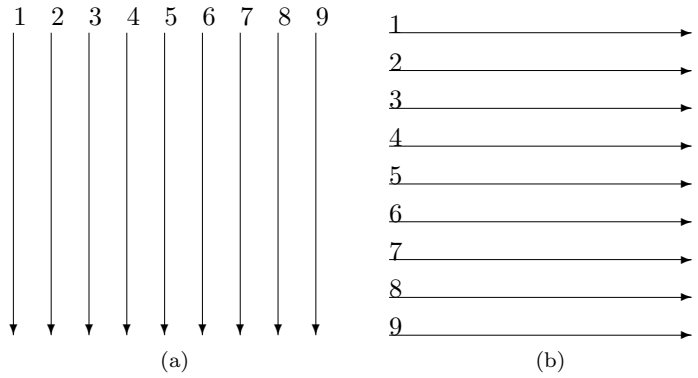


Figure 7: Column- and row-wise pixel ordering. In AOS we apply, for example, TDMA first along all the columns and then along all the rows. (a) Column wise ordering; (b) row wise ordering.

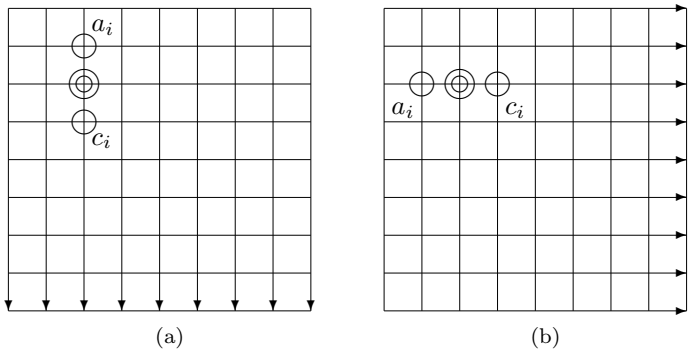


Figure 8: Pixel positions a_i and c_i of b_i, depending on the pixel ordering: (a) column wise ordering; (b) row wise ordering.

5.2 Matlab Implementation

Following is a Matlab implementation, based on the Wikipedia Matlab code[2], of the TDMA algorithm. Since each of the 'columns' (column-wise traversing) and 'rows' (row-wise traversing) are independent of each other, these can be processed is parallel, if needed.

[2] `http://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm`

Algorithm 1 Tridiagonal matrix algorithm. a, b, c are the column vectors for the compressed tridiagonal matrix, d is the right vector.

```
function x = TDMA(a, b, c, d)
    [rows cols dim] = size(a);

    %Modify the first-row coefficients
    c(1, :) = c(1, :) ./ b(1, :);
    d(1, :) = d(1, :) ./ b(1, :);

    %Forward pass
    for i = 2:rows-1
        temp = b(i, :) - a(i, :) .* c(i-1, :);
        c(i, :) = c(i, :) ./ temp;
        d(i, :) = (d(i, :) - a(i, :) .* d(i-1, :)) ./ temp;
    end

    %Backward pass
    x(rows, :) = d(rows, :);
    for i = rows-1:-1:1
        x(i, :) = d(i, :) - c(i, :) .* x(i + 1, :);
    end
end function
```

6 Acknowledgements

I would like to thank everyone who helped me reviewing my PhD thesis (on which this text is based on). Especially, I would like to thank Dr. Florian Pokorny and Dr. Karl Pauwels for their help with the mathematical notation (which can become quite clumsy) and Pablo Guzmán for reviewing this particular version of the text.

References

[1] T. Brox. *From Pixels to Regions: Partial Differential Equations in Image Analysis*. PhD thesis, Saarland University, Saarbrücken, Germany, 2005.

[2] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:629–639, 1990.

[3] J. Ralli. *Fusion and Regularisation of Image Information in Variational Correspondence Methods*. PhD thesis, University of Granada, Spain, 2011.

[4] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.

[5] J. Weickert, B. M. Ter Haar Romeny, and M. A. Viergever. Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Transactions on Image Processing*, 7:398–410, 1998.