5

PARALLEL CO-VOLUME SUBJECTIVE SURFACE METHOD FOR 3D MEDICAL IMAGE SEGMENTATION

Karol Mikula

Department of Mathematics and Descriptive Geometry, Slovak University of Technology, Bratislava, Slovakia

Alessandro Sarti

Dipartmento di Elettronica, Informatica e Sistemistica, University of Bologna, Bologna, Italy

In this chapter we present a parallel computational method for 3D image segmentation. It is based on a three-dimensional semi-implicit complementary volume numerical scheme for solving the Riemannian mean curvature flow of graphs called the subjective surface method. The parallel method is introduced for massively parallel processor (MPP) architecture using the message passing interface (MPI) standard, so it is suitable, e.g., for clusters of Linux computers. The scheme is applied to segmentation of 3D echocardiographic images.

1. INTRODUCTION

The aim of segmentation is to find boundaries of an object in an image. In a generic situation these boundaries correspond to edges. In the presence of noise, which is intrinsically linked to modern noninvasive acquisition techniques (such as ultrasound), the object boundaries (image edges) can be very irregular or even interrupted. The same happens in images with occlusions, subjective contours (in

Address all correspondence to: Karol Mikula, Department of Mathematics and Descriptive Geometry, Slovak University of Technology, Radlinsk,ho 11, 813 68 Bratislava, Slovakia. Phone: 02 5292 5787; Fax: 02 5292 5787. mikula@vox.svf.stuba.sk

some psychologically motivated examples), or in case of partly missing information (like in problems of inpainting). Then simple segmentation techniques fail and image analysis becomes a difficult task. In all these situations, the subjective surface method can help significantly. It is an evolutionary method based on numerical solution of time-dependent highly nonlinear partial differential equations (PDEs), solving a Riemannian mean curvature flow of a graph problem. The segmentation result is obtained as a "steady state" of this evolution. In case of a large dataset (3D images or image sequences), where the ammount of processed information is huge, a discretization of the partial differential equation leads to systems of equations with a huge amount of unknowns. Then parallel implementation of the method is necessary, first, due to a large memory requirement, and second, due to a necessity for fast computing times. For both purposes, an implementation on the massively parallel processor (MPP) architecture using the message passing interface (MPI) standard is a favourable solution.

2. MATHEMATICAL MODELS IN IMAGE SEGMENTATION

Image segmentation based on the subjective surface method is related to geodesic (or conformal) mean curvature flow of level sets (level curves in case of 2D images and level surfaces in case of 3D images). Let us outline first the curve and surface evolution models and their level set formulations, preceding the subjective surface method.

A simple approach to image segmentation (similar to various discrete region-growing algorithms) is to place a small seed, e.g., a small circle in the 2D case, or a small ball in the 3D case, inside the object and then evolving this *segmentation curve* or *segmentation surface* to find automatically the object boundary (cf. [1]). Complex mathematical models as well as Lagrangean numerical schemes have been suggested and studied for evolving curves and surfaces over the last two decades (see, e.g., [2, 3, 4, 5, 6, 7, 8, 9, 10]). For moving curves and surfaces the robust level set models and methods were introduced (see, e.g., [11, 12, 13, 14, 15, 16, 17, 18]). A basic idea in the level set methods is that the moving curve or surface corresponds to the evolution of a particular level curve or level surface of the so-called level set function u that solves some form of the following general level set equation: $u_t = F|\nabla u|$, where F represents the normal component of the velocity of this motion.

The first segmentation level set model with the speed of the segmentation curve (surface) modulated by $F = g^0 \equiv g(|\nabla G_\sigma * I^0|)$, where G_σ is a smoothing kernel and g is a smooth edge detector function, e.g., $g(s) = 1/(1 + Ks^2)$, and was given in [19] and [20]. Due to the shape of the Perona-Malik function g, the moving curve is strongly slowed down in a neighbourhood of an edge, and a "steady state" of the segmentation curve is taken as the boundary of segmented object. However, if an edge is crossed during evolution (e.g., in a noisy image),

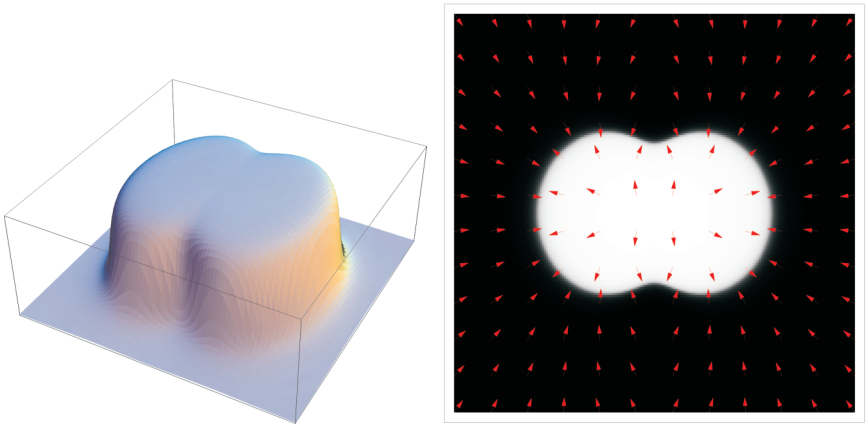

Figure 1. Left: A graph of the image intensity function $I^0(x)$. **Right**: Image given by the intensity $I^0(x)$ plotted together with arrows representing the vector field $-\nabla g(|\nabla I^0(x)|)$. See attached CD for color version.

there is no mechanism to reverse the motion since F is always positive. Moreover, if there is a missing part of the object boundary, such an algorithm, as with any other simple region-growing method, is completely useless.

Not only the segmentation models but also image smoothing (filtering) models and methods have been suggested either using the original Perona-Malik idea of nonlinear diffusion depending on an edge indicator (cf. [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34]) or using geometrical PDEs in the level set formulations (cf. [35, 36, 37, 38, 16, 39, 40, 41, 42]).

Later on, the level set models for image segmentation were significantly improved by introducing a driving force in the form $-\nabla g(|\nabla I^0(x)|)$ ([43, 44, 45, 46, 47]). The vector field $-\nabla g(|\nabla I^0(x)|)$ has an important geometric property: it points toward regions where the norm of the gradient ∇I^0 is large (see Figure 1, illustrating the 2D situation). If an initial segmentation curve or surface belongs to a neighborhood of an edge, it is driven automatically to this edge by the velocity field.

However, the situation is more complicated in the case of noisy images (see Figure 2). The advection is not sufficient, the evolving curve can be attracted to spurious edges, and no reasonably convergent process is observed. Adding a curvature dependence (regularization) to the normal velocity F, the sharp curve irregularities are smoothed, as presented in the right-hand part of Figure 2. It turns out that an appropriate regularization term is given by $g^0 k$, where the amount of curve intrinsic diffusion is small in the vicinity of an un-spurious edge. Following this 2D example, we can write the geometrical equation for the normal velocity v

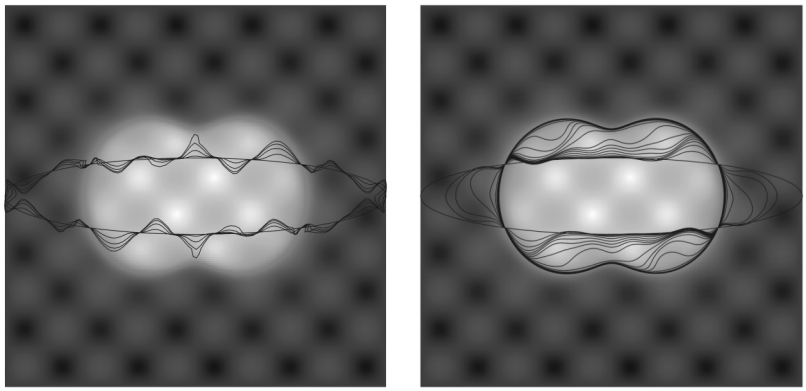

Figure 2. Evolution only by advection leads to attracting a curve (initial ellipse) to spurios edges, but adding a regularization term related to the curvature of evolving curve, the edge is found smoothly also in the case of a 2D noisy image (right).

as

$$v = g^0 k + \nabla g^0 \cdot \vec{N}$$

of the segmentation curve, where k is its curvature and $\vec{N}$ is its normal vector. Similarly, the geometrical equation for the moving segmentation surface has the form

$$v = g^0 H + \nabla g^0 \cdot \vec{N},$$

where H is its mean curvature and $\vec{N}$ is its normal vector. The level set formulation of either such curve or surface evolution is given by ([43, 44, 45, 46, 47])

$$u_t = g^0 |\nabla u| \nabla . \left(\frac{\nabla u}{|\nabla u|} \right) + \nabla g^0 . \nabla u = |\nabla u| \nabla . \left(g^0 \frac{\nabla u}{|\nabla u|} \right), \qquad (1)$$

where the moving curve or surface is given by the same evolving level line and, respectively, level surface of the level set function u.

There is still a practical problem with the previous approach. It gives satisfactory results if the initial segmentation curve or surface belongs to the vicinity of an edge; otherwise, it is difficult to drive an arbitrary initial state there. An important observation, leading to the subjective surface method ([48, 49, 50]), is that Eq. (1) moves not only one particular level set, but all the level sets, by the above mentioned advection-diffusion mechanism. So we can consider the evolution of the whole (hyper)surface u, which we call the *segmentation function*, composed by those level sets. Moreover, we are a bit free in choosing the precise form of the

diffusion term in the segmentation model. In fact,

$$u_t = \sqrt{\varepsilon^2 + |\nabla u|^2}\nabla \cdot \left(g(|\nabla G_\sigma * I^0|) \frac{\nabla u}{\sqrt{\varepsilon^2 + |\nabla u|^2}} \right), \tag{2}$$

where the Evans-Spruck regularization [51]

$$|\nabla u| \approx |\nabla u|_\varepsilon = \sqrt{\varepsilon^2 + |\nabla u|^2} \tag{3}$$

is used, gives the same advection term $-\nabla g^0 \cdot \nabla u$ as (1). The parameter ε shifts the model from the mean curvature motion of level sets ($\varepsilon = 0$) to the mean curvature flow of graphs ($\varepsilon = 1$). This means that either level sets of the segmentation function move in the normal direction proportionally to the (mean) curvature ($\varepsilon = 0$), or the graph of the segmentation function itself moves (as a 2D surface in 3D space in segmentation of 2D images, or a 3D hypersurface in 4D space in segmentation 3D images) in the normal direction proportionally to the mean curvature. In both cases large variations in the graph of the segmentation function outside edges are smoothed due to large mean curvature. On edges the advection dominates, so all the level sets that are close to the edge are attracted from both sides to this edge and a shock (steep gradient) is subsequently formed. For example, if the initial "point-of-view" surface, as plotted in the top right portion of Figure 3, illustrating the 2D situation, is evolved by Eq. (2), the so-called subjective surface is formed finally (see Figure 3, bottom right), and it is easy to use one of its level lines, e.g., $(\max(u) + \min(u))/2$, to get the boundary of the segmented object.

In the next example we illustrate the role of the regularization parameter ε. The choice of $\varepsilon = 1$ is not appropriate for segmentation of an image object with a gap, as seen in Figure 4 (top). However, decreasing ε, i.e., if we go closer to the level set flow Eq. (1), we get very good segmentation results for that image containing a circle with a large gap, as presented in Figure 4 (middle and bottom). If the image is noisy, the motion of the level sets to the shock is more irregular, but finally the segmentation function is smoothed and flattened as well. For a comprehensive overview of the role of all the model parameters, the reader is referred, e.g., to [52].

The subjective surface segmentation (2) is accompanied by Dirichlet boundary conditions:

$$u(t, x) = u^D \quad \text{in } [0, T] \times \partial\Omega, \tag{4}$$

where $\partial\Omega$ is a Lipschitz continuous boundary of a computational domain $\Omega \subset I\!R^d$, $d = 3$, and with initial condition

$$u(0, x) = u^0(x) \quad \text{in } \Omega. \tag{5}$$

We assume that the initial state of the segmentation function is bounded, i.e., $u^0 \in L_\infty(\Omega)$. The segmentation is an evolutionary process given by the solution

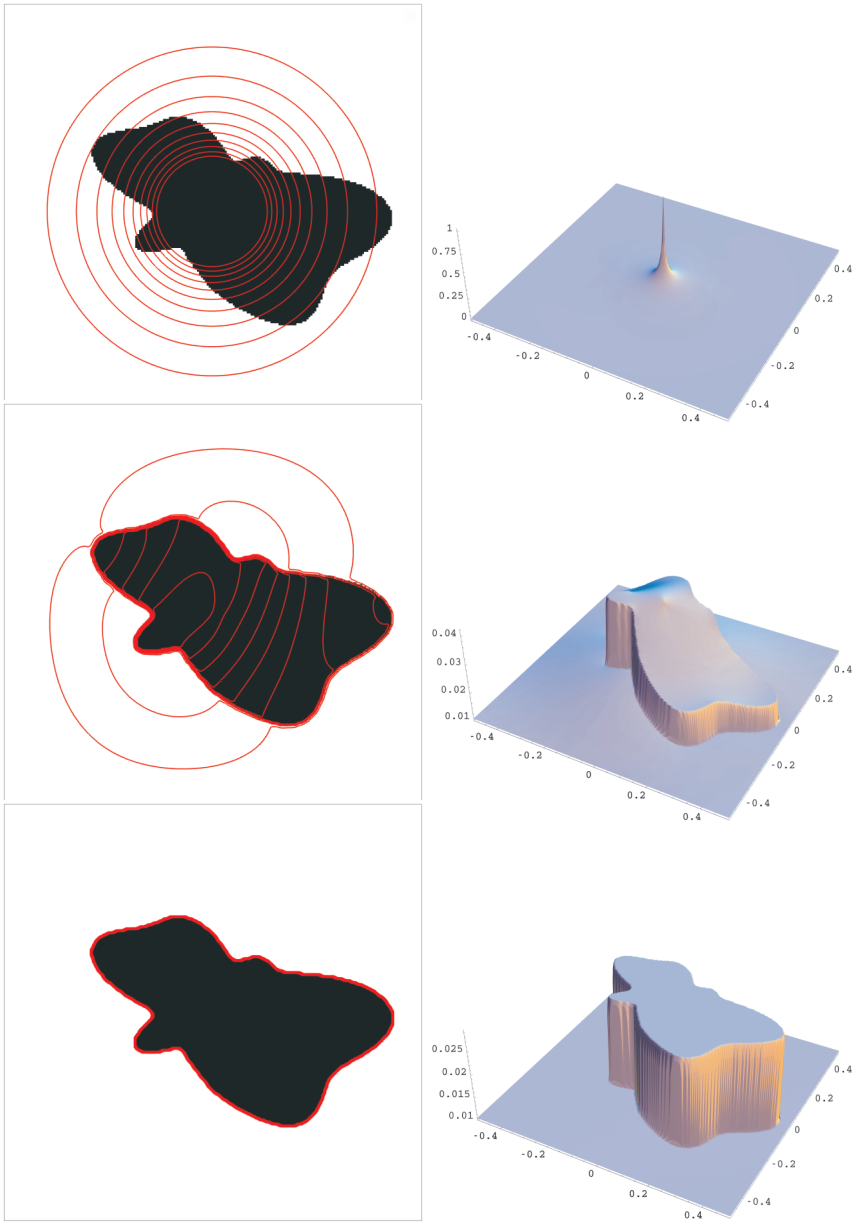

Figure 3. Subjective surface-based segmentation of a "batman" image. In the left column we plot the black-and-white image to be segmented together with isolines of the segmentation function. In the right column there is the shape of the segmentation function. The rows correspond to time steps 0, 1, and 10, which gives the final result. The regularization parameter $\varepsilon = 1$ is used in this example. See attached CD for color version.

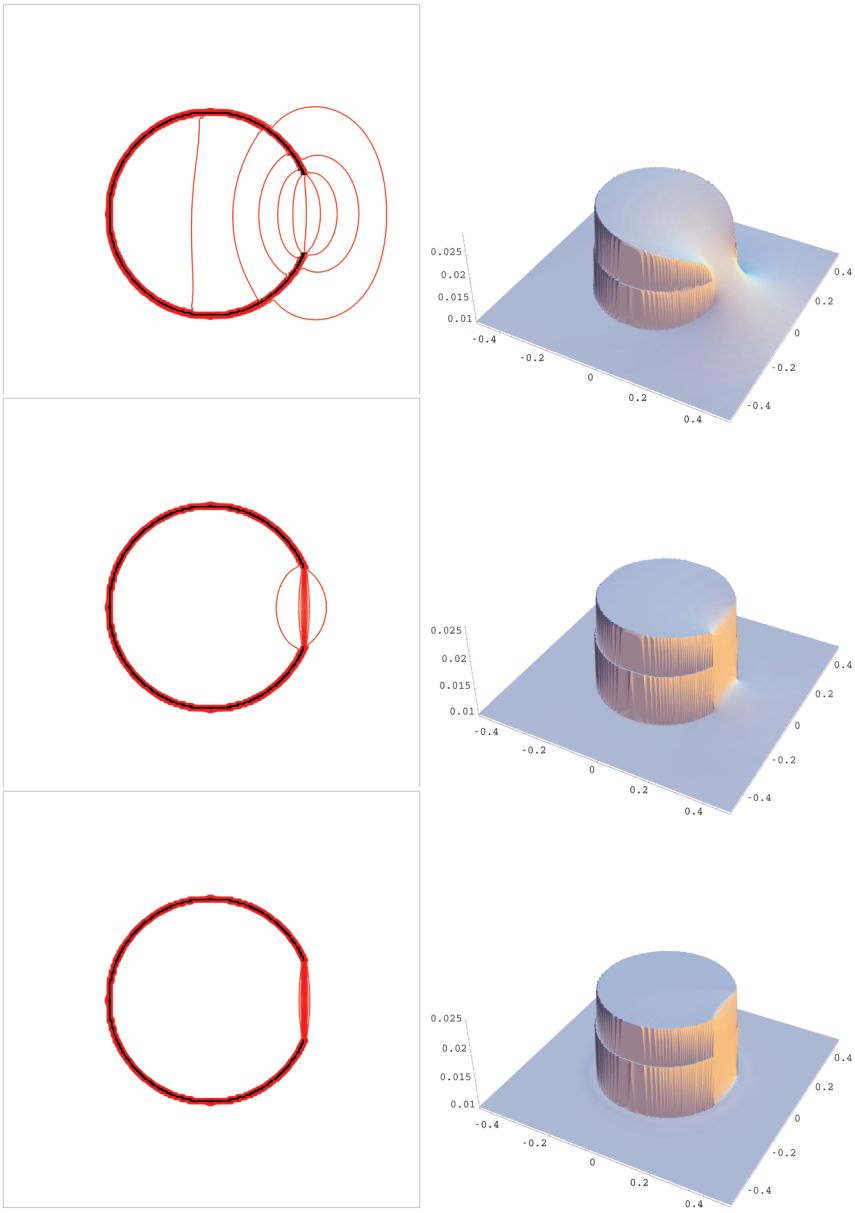

Figure 4. Segmentation of the circle with a big gap using $\varepsilon = 1$ (top), $\varepsilon = 10^{-2}$ (middle), and $\varepsilon = 10^{-5}$ (bottom). For a bigger missing part, a smaller ε is desirable. In the left column we see how closely to the edges the isolines are accumulating and closing the gap; on the right we see how steep the segmentation function is along the gap. See attached CD for color version.

of Eq. (2), where T represents a time when a segmentation result is achieved. The Perona-Malik function $g : \mathbb{R}_0^+ \to \mathbb{R}^+$ is nonincreasing, $g(0) = 1$, admitting $g(s) \to 0$ for $s \to \infty$ [21]. Usually we use the function $g(s) = 1/(1 + Ks^2)$, $K \geq 0$. $G_\sigma \in C^\infty(\mathbb{R}^d)$ is a smoothing kernel, e.g., the Gauss function

$$G_\sigma(x) = \frac{1}{(4\pi\sigma)^{d/2}} e^{-|x|^2/4\sigma}, \tag{6}$$

which is used in pre-smoothing of image gradients by the convolution

$$\nabla G_\sigma * I^0 = \int_{\mathbb{R}^d} \nabla G_\sigma(x - \xi)\tilde{I}^0(\xi)d\xi, \tag{7}$$

with $\tilde{I}^0$ the extension of I^0 to $\mathbb{R}^d$ given by periodic reflection through the boundary of the image domain. The computational domain Ω is usually a subdomain of the image domain, and it should include the segmented object. In fact, in most situations Ω corresponds to the image domain itself. Due to the properties of function g and the smoothing effect of convolution, we always have $1 \geq g^0 \geq \nu_\sigma > 0$ [22, 24]. In [51, 53], the existence of a viscosity solution [54] of the curvature driven level set equation [11], i.e., Eq. (1) with $g^0 \equiv 1$, was proven. For analytical results on Eqs. (1) and (2), respectively, we refer the reader to [47, 45] and [49, 55], respectively.

3. SEMI-IMPLICIT 3D CO-VOLUME SCHEME

Our computational method for solving the subjective surface segmentation equation (2) uses an efficient and unconditionally stable semi-implicit time discretization, first introduced for solving level set-like problems in [16], and a three-dimensional complementary volume spatial discretization introduced in [56] for image processing applications. In this section we present the serial algorithm of our method, and in the next section we introduce its parallel version suitable for a massively parallel computer architecture using the message passing interface standard.

For time discretization of nonlinear diffusion equations there are basically three possibilities: implicit, semi-implicit, or explicit schemes. For spatial discretization usually finite differences [13, 14], finite volumes [57, 58, 59, 26], or finite-element methods [60, 61, 62, 63, 64, 30, 24] are used. The co-volume technique (also called the complementary volume or finite volume-element method) is a combination of the finite-element and finite-volume methods. The discrete equations are derived using the *finite-volume methodology*, i.e., integrating an equation into the so-called control (complementary, finite) volume. Very often the

control volumes are constructed as elements of a dual (complementary) grid to a *finite-element triangulation* (tetrahedral grid in the 3D case). Then the nonlinear quantities in PDEs, as an absolute value of the solution gradient in Eq. (2), are evaluated using piecewise linear representation of the solution on a tetrahedral grid thus employing the methodology of the linear finite-element method. The finite-volume methodology brings in the naturally discrete minimum–maximum principle. The piecewise linear representation (reconstruction) of the segmentation function on the finite-element grid yields a fast and simple evaluation of nonlinearities. Implicit, i.e., nonlinear time discretization and co-volume techniques, for solution of the level set equations were first introduced in [65]. The implicit time stepping as in [65], although unconditionally stable, leads to solution of a nonlinear system in every discrete time update. On the other hand, the semi-implicit scheme leads in every time step to solution of a linear algebraic system that is much more efficient. Using explicit time stepping, stability is often achieved only under severe time step restriction. Since in nonlinear diffusion problems (like the level set equations or the subjective surface method) the coefficients depend on the solution itself and thus must be recomputed in every discrete time update, an overall computational time for an explicit scheme can be tremendous. From such a point of view, the semi-implicit method seems to be optimal regarding stability and computational efficiency.

In the next subsections we discuss the semi-implicit 3D co-volume method. We present the method formally in discretization of Eq. (1), although we always use its ε-regularization (2) with a specific $\varepsilon > 0$. The notation is simpler in case of (1), and it will be clear where the ε-regularization appears in the numerical scheme.

3.1. Semi-Implicit Time Discretization

We first choose a uniform discrete time step τ and a variance σ of the smoothing kernel G_σ. We then replace the time derivative in (1) by backward difference. The nonlinear terms of the equation are treated from the previous time step while the linear ones are considered on the current time level, which means semi-implicitness of the time discretization. By such an approach we get our semi-discrete in a time scheme:

Let τ and σ be fixed numbers, I^0 be a given image, and u^0 a given initial segmentation function. Then, for every discrete time moment $t_n = n\tau$, $n = 1, \dots N$, we look for a function u^n, the solution of the equation

$$\frac{1}{|\nabla u^{n-1}|}\frac{u^n - u^{n-1}}{\tau} = \nabla . \left(g^0 \frac{\nabla u^n}{|\nabla u^{n-1}|} \right). \tag{8}$$

3.2. Co-Volume Spatial Discretization in 3D

A 3D digital image is given on a structure of voxels with a cubic shape, in general. Since discrete values of image intensity I^0 are given in voxels and they influence the model, we will relate spatially discrete approximations of the segmentation function u also to the voxel structure; more precisely, to voxel centers. In every discrete time step t_n of the method (8) we have to evaluate the gradient of the segmentation function at the previous step $|\nabla u^{n-1}|$. To that goal we put the 3D tetrahedral grid into the voxel structure and take a piecewise linear representation of the segmentation function on such a grid. Such an approach will give a constant value of the gradient in tetrahedra (which is the main feature of the co-volume [65, 16] and linear finite-element [62, 63, 64] methods in solving the mean curvature flow in the level set formulation), allowing simple, clear, and fast construction of a fully-discrete system of equations.

The formal construction of our co-volumes will be given in the next paragraph, and we will see that the co-volume mesh corresponds back to the image voxel structure, which is reasonable in image processing applications. On the other hand, the construction of the co-volume mesh has to use a 3D tetrahedral finite-element grid to which it is complementary. This will be possible using the following approach. First, every cubic voxel is split into 6 pyramids with a vertex given by the voxel center and base surfaces given by the voxel boundary faces. The neighbouring pyramids of the neighbouring voxels are joined together to form an octahedron that is then split into 4 tetrahedra using diagonals of the voxel boundary face (see Figure 5). In such way we get our *3D tetrahedral grid*. Two nodes of every tetrahedron correspond to the centers of neighbouring voxels, and the further two nodes correspond to the voxel boundary vertices; every tetrahedron intersects a common face of neighbouring voxels. In our method, only the centers of the voxels will represent degree-of-freedom nodes (DF nodes), i.e., solving the equation at a new time step, we update the segmentation function only in these DF nodes. Additional nodes of the tetrahedra will not represent degrees of freedom, and we will call them non-degree-of-freedom nodes (NDF nodes), and they will be used in piecewise linear representation of the segmentation function. Let a function u be given by discrete values in the voxel centers, i.e., in DF nodes. Then in the NDF nodes we take the average value of the neighbouring DF nodal values. By such defined values in the NDF nodes a piecewise linear approximation u_h of u on the tetrahedral grid is built.

For the tetrahedral grid $\mathcal{T}_h$, given by the previous construction, we construct a co-volume (dual) mesh. We modify the approach given in [65, 16] in such a way that our co-volume mesh will consist of cells p associated only with DF nodes p of $\mathcal{T}_h$, say $p = 1, \ldots, M$. Since there will be one-to-one correspondence between co-volumes and DF nodes, without any confusion, we use the same notation for them. For each DF node p of $\mathcal{T}_h$, let C_p denote the set of all DF nodes q connected to the node p by an edge. This edge will be denoted by σ_{pq} and its length by h_{pq}. Then

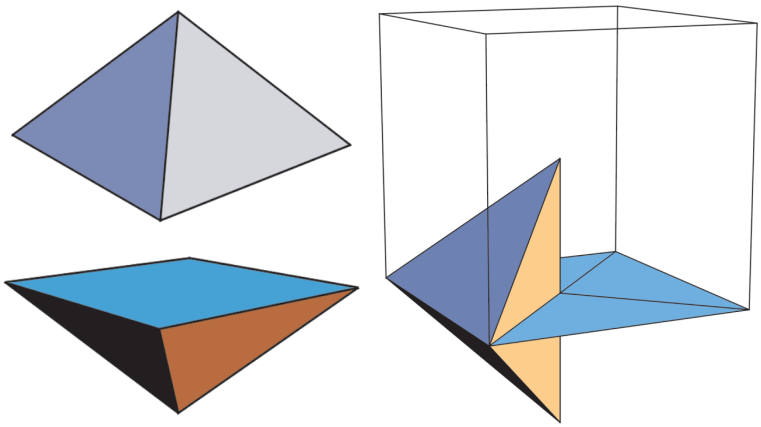

Figure 5. Neighbouring pyramids (left) that are joined together and which, after splitting into four parts, give tetrahedra of our 3D grid. We can the see intersection of one of these tetrahedra with the bottom face of the voxel co-volume (right). See attached CD for color version.

every *co-volume* p is bounded by the planes e_{pq} that bisect and are perpendicular to the edges $\sigma_{pq}, q \in C_p$. By this construction, if e_{pq} intersects σ_{pq} in its center, the co-volume mesh corresponds exactly to the voxel structure of the image inside the computational domain Ω where the segmentation is provided. Then the co-volume boundary faces do cross in NDF nodes. So we can also say that the NDF nodes correspond to zero-measure co-volumes and thus do not add additional equations to the discrete model (cf. (10)), and they do not represent degrees of freedom in the co-volume method. We denote by $\mathcal{E}_{pq}$ the set of tetrahedra having σ_{pq} as an edge. In our situation (see Figure 4), every $\mathcal{E}_{pq}$ consists of 4 tetrahedra. For each $T \in \mathcal{E}_{pq}$, let c_{pq}^T be the area of the portion of e_{pq} that is in T, i.e., $c_{pq}^T = m(e_{pq} \cap T)$, where m is a measure in $I\!R^{d-1}$. Let $\mathcal{N}_p$ be the set of all tetrahedra that have a DF node p as a vertex. Let u_h be a piecewise linear function on $\mathcal{T}_h$. We will denote a constant value of $|\nabla u_h|$ on $T \in \mathcal{T}_h$ by $|\nabla u_T|$ and define regularized gradients by

$$|\nabla u_T|_\varepsilon = \sqrt{\varepsilon^2 + |\nabla u_T|^2}. \tag{9}$$

We will use the notation $u_p = u_h(x_p)$, where x_p is the coordinate of the DF node p of $\mathcal{T}_h$.

With these notations, we are ready to derive co-volume spatial discretization. As is usual in finite-volume methods [59, 58, 57], we integrate ((8)) over every co-volume $p, \imath = 1, \ldots, M$. We get

$$\int_p \frac{1}{|\nabla u^{n-1}|} \frac{u^n - u^{n-1}}{\tau} dx = \int_p \nabla . \left(g^0 \frac{\nabla u^n}{|\nabla u^{n-1}|} \right) dx. \tag{10}$$

For the right-hand side of (10) using the divergence theorem we get

$$\begin{aligned}\int_p \nabla \cdot \left(g^0 \frac{\nabla u^n}{|\nabla u^{n-1}|} \right) dx &= \int_{\partial p} \frac{g^0}{|\nabla u^{n-1}|} \frac{\partial u^n}{\partial \nu} ds \\ &= \sum_{q \in C_p} \int_{e_{pq}} \frac{g^0}{|\nabla u^{n-1}|} \frac{\partial u^n}{\partial \nu} ds.\end{aligned}$$

So we have an integral formulation of (8):

$$\int_p \frac{1}{|\nabla u^{n-1}|} \frac{u^n - u^{n-1}}{\tau} dx = \sum_{q \in C_p} \int_{e_{pq}} \frac{g^0}{|\nabla u^{n-1}|} \frac{\partial u^n}{\partial \nu} ds, \tag{11}$$

expressing a "local mass balance" in the scheme. Now the exact "fluxes" $\int_{e_{pq}} \frac{g^0}{|\nabla u^{n-1}|} \frac{\partial u^n}{\partial \nu} ds$ on the right-hand side and the "capacity function" $\frac{1}{|\nabla u^{n-1}|}$ on the left-hand side (see, e.g., [59]) will be approximated numerically using piecewise linear reconstruction of u^{n-1} on the tetrahedral grid $\mathcal{T}_h$. If we denote by g^0_T the approximation of g^0 on a tetrahedron $T \in \mathcal{T}_h$, then for the approximation of the right-hand side of (11) we get

$$\sum_{q \in C_p} \left(\sum_{T \in \mathcal{E}_{pq}} c^T_{pq} \frac{g^0_T}{|\nabla u^{n-1}_T|} \right) \frac{u^n_q - u^n_p}{h_{pq}}, \tag{12}$$

and the left-hand side of (11) is approximated by

$$M_p m(p) \frac{u^n_p - u^{n-1}_p}{\tau}, \tag{13}$$

where $m(p)$ is a measure in $\mathbb{R}^d$ of co-volume p and M_p is an approximation of the capacity function inside the finite volume p. For that goal we use the averaging of the gradients in tetraherda crossing co-volume p, i.e.,

$$M_p = \frac{1}{|\nabla u^{n-1}_p|}, \quad |\nabla u^{n-1}_p| = \sum_{T \in \mathcal{N}_p} \frac{m(T \cap p)}{m(p)} |\nabla u^{n-1}_T|. \tag{14}$$

Then the regularization of the capacity function is given by

$$M^\varepsilon_p = \frac{1}{|\nabla u^{n-1}_p|_\varepsilon}, \tag{15}$$

and if we define coefficients (where the ε-regularization is taken into account),

$$d^{n-1}_p = M^\varepsilon_p m(p), \tag{16}$$

$$a^{n-1}_{pq} = \frac{1}{h_{pq}} \sum_{T \in \mathcal{E}_{pq}} c^T_{pq} \frac{g^0_T}{|\nabla u^{n-1}_T|_\varepsilon}, \tag{17}$$

we get from (12)–(13) our **3D fully-discrete semi-implicit co-volume scheme**:

Let u_p^0, $p = 1, \ldots, M$ be given discrete initial values of the segmentation function. Then, for $n = 1, \ldots, N$ we look for $u_p^n, p = 1, \ldots, M$, satisfying

$$d_p^{n-1}\, u_p^n + \tau \sum_{q \in C_p} a_{pq}^{n-1}(u_p^n - u_q^n) = d_p^{m-1}\, u_p^{n-1}. \tag{18}$$

The system (18) can be rewritten into the form

$$\left(d_p^{n-1} + \tau \sum_{q \in C_p} a_{pq}^{n-1} \right) u_p^n - \tau \sum_{q \in C_p} a_{pq}^{n-1} u_q^n = d_p^{n-1}\, u_p^{n-1}, \tag{19}$$

and, applying the Dirichlet boundary conditions (which contribute to the right-hand side), it gives a system of linear equations with a matrix $\mathbf{A}_{M\times M}$, the off-diagonal elements of which are symmetric and nonpositive, namely $\mathbf{A}_{pq} = -\tau a_{pq}^{n-1}$, $q \in C_p$, $\mathbf{A}_{pq} = 0$, otherwise. Diagonal elements are positive, namely, $\mathbf{A}_{pp} = d_p^{n-1} + \tau \sum\limits_{q \in C_p} a_{pq}^{n-1}$, and dominate the sum of the absolute values of the nondiagonal elements in every row. Thus, the matrix of the system is symmetric and a diagonally dominant M-matrix, which implies that it always has a unique solution for any $\tau > 0$, $\varepsilon > 0$, and for every $n = 1, \ldots, N$. The M-matrix property gives us the minimum–maximum principle:

$$\min_p\, u_p^0 \le \min_p\, u_p^n \le \max_p\, u_p^n \le \max_p\, u_p^0,\ \ 1 \le n \le N, \tag{20}$$

which can be seen by the following simple trick. We may temporary rewrite (18) into the equivalent form:

$$u_p^n + \frac{\tau}{d_p^{n-1}} \sum_{q \in C_p} a_{pq}^{n-1}(u_p^n - u_q^n) = u_p^{n-1}, \tag{21}$$

and let $\max(u_1^n, \ldots, u_M^n)$ be achieved in the node p. Then the whole second term on the left-hand side is nonnegative, and thus $\max(u_1^n, \ldots, u_M^n) = u_p^n \le u_p^{n-1} \le \max(u_1^{n-1}, \ldots, u_M^{n-1})$. In the same way, we can prove the relation for the minimum, and together we have

$$\min_p\, u_p^{n-1} \le \min_p\, u_p^n \le \max_p\, u_p^n \le \max_p\, u_p^{n-1},\ \ 1 \le n \le N, \tag{22}$$

which by recursion implies the L_∞ stability estimate (20).

The evaluation of g_T^0 included in coefficients (17) can be done in several ways. First, we may replace the convolution by the weighted average to get $I_\sigma^0 := G_\sigma * I^0$ (see, e.g., [26]), and then relate discrete values of I_σ^0 to voxel centers. Then, as above, we may construct its piecewise linear representation on the grid and get a constant value of $g_T^0 \equiv g(|\nabla I_\sigma^0|)$ on every tetrahedron $T \in \mathcal{T}_h$. Another possibility is to solve numerically the linear heat equation for time t corresponding to variance σ with the initial datum given by I^0 (see, e.g., [30]) by the same method as above. The convolution represents a preliminary smoothing of the data. It is also a theoretical tool to have bounded gradients and thus a strictly positive weighting coefficient g^0. In practice, the evaluation of gradients on a fixed discrete grid (e.g., described above) always gives bounded values. So, working on a fixed grid, one can also avoid the convolution, especially if preliminary denoising is not needed or not desirable. Then it is possible to work directly with gradients of the piecewise linear representation of I^0 in evaluation of g_T^0.

A change in the L_2 norm of numerical solutions in subsequent time steps is used to stop the segmentation process. We check whether

$$\sqrt{\sum_p m(p)\ (u_p^n - u_p^{n-1})^2} < \delta, \tag{23}$$

with a prescribed threshold δ. For our semi-implicit scheme and small ε, a good choice of threshold is $\delta = 10^{-5}$.

We start all computations with an initial function given as a peak centered in a "focus point" inside the segmented object. Such a function can be described at a sphere with center s and radius R by $u^0(x) = \frac{1}{|x-s|+v}$, where s is the focus point and $\frac{1}{v}$ gives a maximum of u^0. Outside the sphere we take the value u^0 equal to $\frac{1}{R+v}$. R usually corresponds to the halved inner diameter of the image domain. For small objects, a smaller R can be used to speed up computations. Usually we put the focus point s inside a small neighborhood of a center of the mass of the segmented object.

3.3. Semi-Implicit 3D Co-Volume Scheme in Finite-Difference Notation

The presented co-volume scheme is designed for the specific mesh given by the cubic voxel structure of a 3D image. For simplicity of implementation, reader convenience, and due to the relation to the next section devoted to parallelization, we will write the co-volume scheme (18) in a "finite-difference notation." As is usual for 3D rectangular grids, we associate co-volume p and its center (DF node) with a triple (i, j, k), i representing the index in the x-direction, j in the y-direction, and k in the z-direction (see Figure 7 for our convention of coordinate notation). The unknown value u_p^n then can be denoted by $u_{i,j,k}^n$. If Ω is a rectangular subdomain of the image domain (usually Ω is the image domain itself) and $N_1 + 1$, $N_2 + 1$, $N_3 + 1$ are the numbers of voxels of Ω in the x, y, z-directions, and if we

consider Dirichlet boundary conditions (i.e., the values u_p^n in boundary voxels are not considered as unknown), then $i = i_l, \ldots, i_r$, $j = j_l, \ldots, j_r$, $k = k_l, \ldots, k_r$, where $i_r - i_l \leq N_1 - 2$, $j_r - j_l \leq N_2 - 2$, $k_r - k_l \leq N_3 - 2$. We define the space discretization step $h = \frac{1}{N_1}$, and for simplicity we assume that voxels have cubic shape. For every co-volume p, the set $C_p = \{w, e, s, n, b, t\}$ consists of 6 neighbours, west $u_{i-1,j,k}$, east $u_{i+1,j,k}$, south $u_{i,j-1,k}$, north $u_{i,j+1,k}$, bottom $u_{i,j,k-1}$, and top $u_{i,j,k+1}$, and the set $\mathcal{N}_p$ consists of 24 tetrahedra.

In every discrete time step $n = 1, \ldots, N$ and for every i, j, k, we compute the absolute value of gradient $|\nabla u_T^{n-1}|$ on these 24 tetrahedra. We denote by $G_{i,j,k}^{z,l}, l = 1, \ldots, 4, z \in C_p$ the square of the gradient on the tetrahedra crossing the west, east, south, north, bottom, and top co-volume faces. If we define (omitting upper index $n - 1$)

$$\begin{aligned} s_{i,j,k} &= (u_{i,j,k} + u_{i-1,j,k} + u_{i,j-1,k} + u_{i-1,j-1,k} + \\ &+ u_{i,j,k-1} + u_{i-1,j,k-1} + u_{i,j-1,k-1} + u_{i-1,j-1,k-1})/8, \end{aligned}$$

the value at the left-south-bottom NDF node of the co-volume, then for the west face we get

$$\begin{aligned} G_{i,j,k}^{w,1} &= \left(\frac{u_{i,j,k} - u_{i-1,j,k}}{h}\right)^2 + \left(\frac{s_{i,j,k+1} - s_{i,j,k}}{h}\right)^2 + \\ &\quad \left(\frac{u_{i,j,k} + u_{i-1,j,k} - s_{i,j,k+1} - s_{i,j,k}}{h}\right)^2, \\ G_{i,j,k}^{w,2} &= \left(\frac{u_{i,j,k} - u_{i-1,j,k}}{h}\right)^2 + \left(\frac{s_{i,j+1,k+1} - s_{i,j,k+1}}{h}\right)^2 + \\ &\quad \left(\frac{s_{i,j+1,k+1} + s_{i,j,k+1} - u_{i,j,k} - u_{i-1,j,k}}{h}\right)^2, \\ G_{i,j,k}^{w,3} &= \left(\frac{u_{i,j,k} - u_{i-1,j,k}}{h}\right)^2 + \left(\frac{s_{i,j+1,k+1} - s_{i,j+1,k}}{h}\right)^2 + \\ &\quad \left(\frac{s_{i,j+1,k+1} + s_{i,j+1,k} - u_{i,j,k} - u_{i-1,j,k}}{h}\right)^2, \\ G_{i,j,k}^{w,4} &= \left(\frac{u_{i,j,k} - u_{i-1,j,k}}{h}\right)^2 + \left(\frac{s_{i,j+1,k} - s_{i,j,k}}{h}\right)^2 + \\ &\quad \left(\frac{u_{i,j,k} + u_{i-1,j,k} - s_{i,j+1,k} - s_{i,j,k}}{h}\right)^2, \end{aligned} \tag{24}$$

and correspondingly we get all $G_{i,j,k}^{z,l}$ for the further co-volume faces.

In the same way, but only once at the beginning of the algorithm, we compute values $G_{i,j,k}^{\sigma,z,l}, l = 1, \ldots, 4, z \in C_p$, changing u by I_σ^0 in the previous expressions, and we apply function g to all these values to get discrete values of g_T^0.

Then in every discrete time step and for every i, j, k we construct (west, east, south, north, bottom and top) coefficients

$$a^w_{i,j,k} = \tau\frac{1}{4}\sum_{l=1}^{4}\frac{g(\sqrt{G^{\sigma,w,l}_{i,j,k}})}{\sqrt{\varepsilon^2 + G^{w,l}_{i,j,k}}},\quad a^e_{i,j,k} = \tau\frac{1}{4}\sum_{l=1}^{4}\frac{g(\sqrt{G^{\sigma,e,l}_{i,j,k}})}{\sqrt{\varepsilon^2 + G^{e,l}_{i,j,k}}},$$

$$a^s_{i,j,k} = \tau\frac{1}{4}\sum_{l=1}^{4}\frac{g(\sqrt{G^{\sigma,s,l}_{i,j,k}})}{\sqrt{\varepsilon^2 + G^{s,l}_{i,j,k}}},\quad a^n_{i,j,k} = \tau\frac{1}{4}\sum_{l=1}^{4}\frac{g(\sqrt{G^{\sigma,n,l}_{i,j,k}})}{\sqrt{\varepsilon^2 + G^{n,l}_{i,j,k}}}, \tag{25}$$

$$a^b_{i,j,k} = \tau\frac{1}{4}\sum_{l=1}^{4}\frac{g(\sqrt{G^{\sigma,b,l}_{i,j,k}})}{\sqrt{\varepsilon^2 + G^{b,l}_{i,j,k}}},\quad a^t_{i,j,k} = \tau\frac{1}{4}\sum_{l=1}^{4}\frac{g(\sqrt{G^{\sigma,t,l}_{i,j,k}})}{\sqrt{\varepsilon^2 + G^{t,l}_{i,j,k}}},$$

and we use, cf. (14),

$$m_{i,j,k} = \frac{1}{\sqrt{\varepsilon^2 + \left(\frac{1}{24}\sum\limits_{z\in C_p}\sum\limits_{l=1}^{4}\sqrt{G^{z,l}_{i,j,k}}\right)^2}}$$

to define diagonal coefficients

$$a^p_{i,j,k} = a^w_{i,j,k} + a^e_{i,j,k} + a^s_{i,j,k} + a^n_{i,j,k} + a^b_{i,j,k} + a^t_{i,j,k} + m_{i,j,k}h^2.$$

If we define the right-hand sides at the nth discrete time step by

$$b_{i,j,k} = m_{i,j,k}h^2 u^{n-1}_{i,j,k},$$

then for the DF node corresponding to triple (i, j, k) we get the equation

$$\begin{aligned} a^p_{i,j,k}u^n_{i,j,k} - a^w_{i,j,k}u^n_{i-1,j,k} - a^e_{i,j,k}u^n_{i+1,j,k} - a^s_{i,j,k}u^n_{i,j-1,k} - \\ a^n_{i,j,k}u^n_{i,j+1,k} - a^b_{i,j,k}u^n_{i,j,k-1} - a^t_{i,j,k}u^n_{i,j,k+1} = b_{i,j,k}. \end{aligned} \tag{26}$$

Collecting these equations for all DF nodes and taking into account Dirichlet boundary conditions we get the linear system to be solved.

3.4. Solution of Linear Systems

We can solve system (26) by any efficient preconditioned linear iterative solver suitable for sparse, symmetric, diagonally dominant M-matrices [66]. For example, the so-called SOR (Successive Over Relaxation) method can be used. Then,

at the nth discrete time step we start the iterations by setting $u_{i,j,k}^{n(0)} = u_{i,j,k}^{n-1}$, $i = i_l, \ldots, i_r$, $j = j_l, \ldots, j_r$, $k = k_l, \ldots, k_r$, and in every iteration $l = 1, \ldots$ and for every $i = i_l, \ldots, i_r$, $j = j_l, \ldots, j_r$, $k = k_l, \ldots, k_r$ the following two-step procedure is used:

$$\begin{aligned} Y &= (a_{i,j,k}^{w} u_{i-1,j,k}^{n(l)} + a_{i,j,k}^{e} u_{i+1,j,k}^{n(l-1)} + a_{i,j,k}^{s} u_{i,j-1,k}^{n(l)} + \\ &\quad a_{i,j,k}^{n} u_{i,j+1,k}^{n(l-1)} + a_{i,j,k}^{b} u_{i,j,k-1}^{n(l)} + a_{i,j,k}^{t} u_{i,j,k+1}^{n(l-1)} + b_{i,j,k}) / a_{i,j,k}^{p} \quad (27) \\ u_{i,j,k}^{n(l)} &= u_{i,j,k}^{n(l-1)} + \omega (Y - u_{i,j,k}^{n(l-1)}). \end{aligned}$$

We define the squared L_2-norm of the residuum after the lth SOR iteration by

$$R^{(l)} = \sum_{i,j,k} (a_{i,j,k}^{p} u_{i,j,k}^{n(l)} - a_{i,j,k}^{w} u_{i-1,j,k}^{n(l)} - a_{i,j,k}^{e} u_{i+1,j,k}^{n(l)} - a_{i,j,k}^{s} u_{i,j-1,k}^{n(l)} - a_{i,j,k}^{n} u_{i,j+1,k}^{n(l)} a_{i,j,k}^{b} u_{i,j,k-1}^{n(l)} - a_{i,j,k}^{t} u_{i,j,k+1}^{n(l)} - b_{i,j,k})^2.$$

The iterative process is stopped if $R^{(l)} < \mathrm{TOL}\, R^{(0)}$. The relaxation parameter ω is chosen by the user to improve the convergence rate of the method.

4. BUILDING UP THE PARALLEL ALGORITHM

4.1. MPI Programming

A parallel computer architecture (cf. [67]) is usually categorized by two aspects: whether the memory is physically centralized or distributed, and whether or not the address space is shared. On one hand there is so-called SMP (symmetric multi-processor) architecture that uses shared system resources, e.g., memory and an input/output subsystem, equally accessible from all processors. On the other hand, there is the MPP (massively parallel processors) architecture, where the so-called nodes are connected by a high-speed network. Each node has its own processor, memory, and input/output subsystem, and the operating system is running on each node. Massively does not necessarily mean a large number of nodes, so one can consider, e.g., a cluster of Linux system computers of a reasonable size (and price) to solve a particular scientific or engineering problem. But, of course, parallel computers with a huge number (hundreds) of nodes are used at large computer centers.

The main goal of parallel programming is to utilize all available processors and minimize the elapsed time of the program. In SMP architecture one can assign the parallelization job to a compiler, which usually parallelizes some DO loops (for image processing applications based on such an approach we refer the reader to, e.g., [68]). This is a simple approach but is restricted to having such memory

resources with a shared address space at one's disposal. In MPP architecture, where the address space is not shared among the nodes, parallel processes must transmit data over a network to access data that other processes update. To that goal, message-passing is employed.

In parallel execution, several programs can cooperate in providing computations and handling data. But our parallel implementation of the co-volume subjective surface algorithm uses so-called the SPMD (single program multiple data) model. In the SPMD model, there is only one program built, and each parallel process uses the same executable working on different sets of data. Since all the processes execute the same program, it is necessary to distinguish between them. To that goal, each process has its own *rank*, and we can let processes behave differently, although executing one program, using the value of *rank*. In our case, we split the huge amount of voxels into several parts, proportional to the number of processors, and then we have to rewrite the serial program in such a way that each parallel process handles the correct part of the data and transmits the necessary information to other processes.

Parallelization should reduce the time spent on computation. If there are p processes involved in parallel execution, ideally, the parallel program could be executed p times faster than a sequential one. However, this is not true in practice because of the necessity for data transmission due to data splitting. This drawback of parallelization can be overcome in an efficient and reliable way by so-called message-passing, which is used to consolidate what has been separated by parallelization.

The Message Passing Interface (MPI) is a standard specifying a portable interface for writing parallel programs that have to utilize the message-passing. It aims at practicality, efficiency, and flexibility at the same time. The MPI subroutines solve the problems of environment management, point-to-point and collective communication among processes, construction of derived data types, input/output operations, etc.

The environment managment subroutines, MPI_Init and MPI_Finalize, initiate and finalize an MPI environment. Using subroutine MPI_Comm_size, one can get a number of processes involved in parallel execution belonging to a *communicator*– identifier associated with a group of processes participating in the parallel job, e.g., MPI_COMM_WORLD. Subroutine MPI_Comm_rank gives a *rank* to a process belonging to communicator. The MPI parallel program should include the file mpi.h, which defines all MPI-related parameters (cf. Figure 6).

Collective communication subroutines allow one to exchange data among a group of processes specified by the communicator, e.g., MPI_Bcast sends data from a specific process called the *root* to all the other processes in the communicator. Or, subroutine MPI_Allreduce does reduction operations such as summation of data distributed over all processes in the communicator and places the result on all of the processes.

```
#include <mpi.h>
int main(int argc, char *arvg[])
{
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
 MPI_Comm_rank(MPI_COMM_WORLD, &myid);
 .
 .
 MPI_Finalize():
}
```

Figure 6. Typical structure of an MPI parallel program.

Using point-to-point communication subroutines, a message is sent by one process and received by another. We distinguish between unidirectional and bidirectional communications. At the sending process, the data are first collected into the *user sendbuffer* (scalar variables or arrays used in the program), and then one of the MPI send subroutines is called, the system copies the data from the user sendbuffer to *system buffer*, and finally the system sends the data from the system buffer to the destination process. During the receiving process, one of the MPI receive subroutines is called, the system receives the data from the source process, and copies it to the *system buffer*, and then the system copies the data from the system buffer to the *user recvbuffer*, and finally the data can be used by the receiving process. In MPI, there are two modes of communication: blocking and non-blocking. Using blocking communication subroutines, the program will not return from the subroutine call until the copy to/from the system buffer has finished. Using non-blocking communication subroutines such as MPI_Isend and MPI_Irecv, the program immediately returns from the subroutine call. This indicates that the copy to/from the system buffer is only initiated, so one has to assure that it is also completed by using the MPI_Wait subroutine. In other cases, incorrect data could be copied to the system buffer. In spite of the higher complexity of non-blocking subroutines, we generally prefer them, because their usage is more safe from *deadlock* in bidirectional communication. Deadlocks can take place either due to the incorrect order of blocking send and receive subroutines or due to the limited size of the system buffer, and when a deadlock occurs, the involved processes will not proceed any further.

In the next subsection we show how the message-passing subroutines mentioned above are employed to parallelize the 3D semi-implicit co-volume subjective surface segmentation method.

4.2. Parallelization of the Co-Volume Algorithm Using MPI

There are two main goals for parallelization of a program: to handle huge amounts of data that cannot be placed into the memory of one single serial computer, and to run the program faster. Let us suppose that in terms of running time, a fraction p of a program can be parallelized. In an ideal situation, executing the parallelized program on n processors, the running time will be $1 - p + \frac{p}{n}$. We can see that, if, e.g., only 80% of the program can be parallelized, i.e., $p = 0.8$, the maximal speed-up (estimated from above by $\frac{1}{1-p}$) cannot exceed 5, although with infinitely many processors. This illustrative example shows that it is very important to identify the fraction of the program that can be parallelized and maximize it. Fortunately, every time-consuming part of our algorithm can be efficiently parallelized either directly (reading and writing data, computing coefficients of the linear system), or it is possible to change the serial linear solver (SOR method) to the parallel solver (e.g., RED-BLACK SOR method), which can be parallelized efficiently. The next important issue in parallelization is to balance the workload of the parallel processes. This problem can be addressed by as uniform as possible splitting of the data so that every process provides approximately the same number of operations. The final and very important step is to minimize the time spent for communication. This leads, e.g., to a requirement that the data transmitted (e.g., multidimensional arrays) be contiguous in memory, so that one can exchange it among processors directly in one message using only one call of MPI send and receive subroutines. For parallel algorithms implemented in C language this means that multidimensional arrays should be distributed in row-wise blocks, or, better, say we have to split the multidimensional array like $u[i][j][k]$ in the first index i (cf. Figure 7).

In Figure 8 we can see the main structure of our parallel program. First, the MPI environment is initiated, and every process involved in parallel execution gets its rank stored in variable $\mathrm{myid} = 0, \ldots, n_{\mathrm{procs}} - 1$, where 0 represents the root process and n_{procs} is the number of parallel processes. Then by the root process we read the time step τ and the upper estimate of the number of time steps nts. These parameters are sent to all processes by the MPI_Bcast subroutine. In the beginning of the algorithm we also read the image, compute the discrete values of g_T^0 in function Coefficients0, and construct the initial segmentation function. All these procedures work independently on their own (overlapping) subsets of data, and no exchange of information between processes is necessary in this part of the program. Then in the cycle we call procedure EllipticStep, which in every time step contains computing of coefficients (25) and solving the linear system (26). In the iterative solution of the linear system we will need to exchange overlapping data between neighbouring processes. The cycle is finished when condition ((23)) is fulfilled, and finally the MPI environment is finalized.

Figure 9 shows our distribution of data among the processes. Both the 3D image and the discrete values of the segmentation function are represented by a

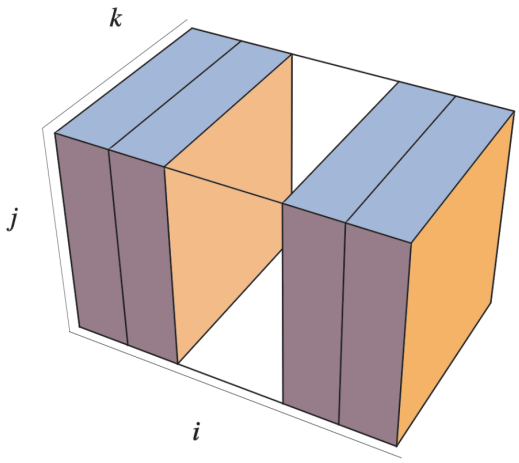

Figure 7. Splitting of a 3D image to n_{procs} 3D rectangular subareas, where n_{procs} corresponds to the number of processes involved in parallel execution. See attached CD for color version.

```
#include <mpi.h>

int main(int argc, char *argv[])
{
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myid);
 if (myid==0)
   {
   printf("tau, number of time steps:\n");
   scanf("%lf %d",&tau,&nts);
   }
 MPI_Bcast(&tau, 1, MPI_DOUBLE,0,MPI_COMM_WORLD);
 MPI_Bcast(&nts, 1, MPI_INT,0,MPI_COMM_WORLD);
 ReadingImage();
 Coefficients0();
 InitialSegmentationFunction();
 for (k=1;k<=nts;k++)
   {
    change=EllipticStep();
    if (change < delta)
      {
       WritingSegmentationResult();
       break;
      }
   }
MPI_Finalize();
}
```

Figure 8. Main structure of our MPI parallel program for the 3D semi-implicit co-volume subjective surface segmentation method.

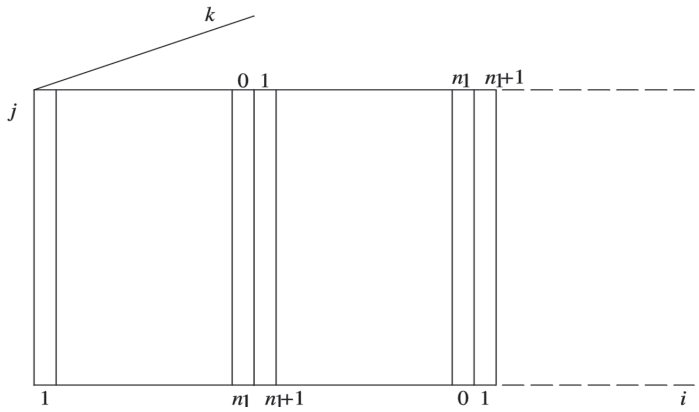

Figure 9. Data distribution and its overlap over parallel processes.

three-dimensional array indexed by i, j, k. The 3D image is given in index ranges $i = 1, \ldots, N_1 + 1$, $j = 1, \ldots, N_2 + 1$, $k = 1, \ldots, N_1 + 3$. Let us suppose that our computational domain, i.e., the domain where we update the segmentation function, is equal to the image domain. Then the boundary positions with $i = 1$, $i = N_1 + 1$, $j = 1$, $j = N_2 + 1$, $k = 1$, $k = N_3 + 1$ are reserved for Dirichlet boundary conditions and all the inner voxel positions correspond to DF nodes of the 3D co-volume algorithm. In order to distribute the data (3D image as well as the segmentation function), we define $n_1 = \frac{N_1}{n_{\text{procs}}} + 1$, $n_1^{\text{last}} = N_1 - (n_{\text{procs}} - 1)n_1$, and we set $n_2 = N_2$, $n_3 = N_3$. Then on the root process with rank 0 we store the first part of the 3D image as well as the first part of the discrete segmentation function, namely, the array with indices $i = 1, \ldots, n_1 + 1$, $j = 1, \ldots, n_2 + 1$, $k = 1, \ldots, n_3+1$ (cf. Figure 9). The next process with rank 1 handles the next part of the image and the segmentation function, namely, all 2D slices $j = 1, \ldots, n_2+1$, $k = 1, \ldots, n_3+1$ locally indexed by i in the range $i = 0, \ldots, n_1+1$, where the 2D slice with index $i = 0$ corresponds to the slice with index $i = n_1$ in the root process (cf. Figure 9). This is similar for further processes, with the only difference that on the last process with rank $n_{\text{procs}} - 1$ the index i of the last 2D slice is n_1^{last} instead of n_1. The merging of all 2D slices for $i = 1, \ldots, n_1$ (n_1^{last} on the last process) from all the subsequent processes gives the non-distributed complete 3D image as well as the complete segmentation function. In order to solve iteratively the linear system and to compute its coefficients, we need the overlap. The overlap in which it is necessary to exchange information between neighbouring processes is given by the slices $n_1, n_1 + 1$ and slices $0, 1$ of subsequent processes (cf. Figure 9).

As a first example showing how the data distribution is realized, we present the procedure for the parallel reading of the 3D image in Figure 10. In all subsequent figures the value of parameter $p = 1$. Depending on the rank of the process, we

```
void ReadingImage()
{
 input=fopen("3Dimage.dat","r");

 for (n=0;n<nprocs;n++)
  {
    if (myid==n)
    {
     if(myid==0)
      {
       fseek(input,0,SEEK_SET);
       for(i=p;i<=n1+p;i++)
       for(j=p;j<=n2+p;j++)
       for(k=p;k<=n3+p;k++)
       {
        ll=getc(input); u[i][j][k]=ll/255.;
       }
      }
    if((myid>0)&&(myid<nprocs-1))
      {
       fseek(input,(n1*myid-1)*(n2+1)*(n3+1),SEEK_SET);
       for(i=p-1;i<=n1+p;i++)
       for(j=p;j<=n2+p;j++)
       for(k=p;k<=n3+p;k++)
       {
        ll=getc(input); u[i][j][k]=ll/255.;
       }
      }
    if (myid==nprocs-1)
      {
       fseek(input,(n1*myid-1)*(n2+1)*(n3+1),SEEK_SET);
       for(i=p-1;i<=n1last+p;i++)
       for(j=p;j<=n2+p;j++)
       for(k=p;k<=n3+p;k++)
       {
        ll=getc(input); u[i][j][k]=ll/255.;
       }
      }
    }
  }
fclose(input);
}
```

Figure 10. Parallel reading of a 3D image.

start the reading of the input file at the desired position and put the graylevel image intensity to the array $0 \leq u[i][j][k] \leq 1$.

After reading of the image, the discrete values of g_T^0 are computed in the procedure Coefficients0 (cf. the paragraph following (24)). Then we do not need an image anymore in the program, so the discrete initial segmentation function is

```
for (i=p+1;i<=N1+p-1;i++)
for (j=p+1;j<=N2+p-1;j++)
for (k=p+1;k<=N3+p-1;k++)
  {
  z=(b[i][j][k]+aw[i][j][k]*u[i-1][j][k]+
     ae[i][j][k]*u[i+1][j][k]+as[i][j][k]*u[i][j-1][k]+
     an[i][j][k]*u[i][j+1][k]+ab[i][j][k]*u[i][j][k-1]+
     at[i][j][k]*u[i][j][k+1])/ap[i][j][k];
  u[i][j][k]=u[i][j][k]+omega*(z-u[i][j][k]);
  }
```

Figure 11. One iteration of the standard serial SOR method.

built and stored in the same array $u[i][j][k]$ by the procedure InitialSegmentation-Function.

In every time step, the values of the discrete segmentation function are updated in the DF nodes, solving iteratively the linear system with coefficients given by (25). Figure 11 shows one iteration of the serial SOR method described in (25). We can see the dependence of the currently updated value $u[i][j][k]$ on its six neighbours (the west, east, south, north, bottom, and top DF nodes). In every iteration three of them should already be known (cf. (27)), so in parallel run every consecutive process should wait until its preceding process is finished in order to get the west values updated. Such dependence is not well suited for parallelization. But there exists an elegant way to change the standard SOR method to be efficiently parallelized (see, e.g., [67]). It is possible to split all voxels to RED elements, given by the condition that the sum of its indices is an even number, and to BLACK elements, given by the condition that sum of its indices is an odd number. Then the six neighbours of RED elements are BLACK elements (cf. Figure 12), and the value of RED elements depends only on those of the BLACK elements, and vice versa. Due to this fact, we can split one SOR iteration into two steps. First we update RED elements and then BLACK elements, and this splitting is perfectly parallelizable. Figure 13 shows one iteration of the so-called RED-BLACK SOR method operating on RED elements.

After computing one RED-BLACK SOR iteration for the RED elements on every parallel process, we have to exchange RED updated values in overlapping regions, and then we can compute one iteration for BLACK elements. The data exchange is implemented as shown in Figure 14 using non-blocking MPI_Isend and MPI_Irecv subroutines. This iterative process is stopped using the condition for a relative residual stated at the end of Section 3.4. Computing the initial residual $R^{(0)}$ as well as all further residuals $R^{(l)}$, we have to collect partial information from all the processes and send the collected value to all processes to check the stopping criterion by every process. Figure 15 shows how the MPI_Allreduce subroutine is used toward that end in computing $R^{(0)}$. In fact, we compute by the

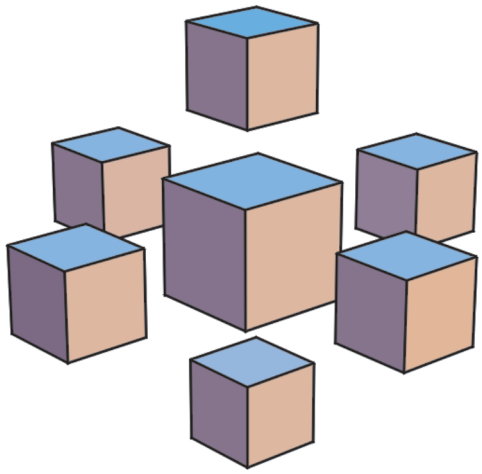

Figure 12. The RED element is in the middle, and its 6 neighbors (west, east, south, north, bottom, and top) have the sum of indices different by 1 from the middle one, so they are all BLACK elements. See attached CD for color version.

same strategy the residuals $R^{(l)}$ not after every RED-BLACK iteration, since it is time consuming in itself, but after every ten RED-BLACK iterations. Fulfiling the stopping criterion for the SOR iterations, we get an approximate solution in the new time step. In checking the stopping condition (23) of the overall segmentation process, we have to employ in a similar way the procedure MPI_Allreduce in evaluation of the L_2 norm of difference of subsequent time step solutions.

Using the PARAVER software, in Figures 16–19 we visualize the run of our parallel program, computing just one time step of the method on four processors. In Figure 16 we can see in green color the time spent for the MPI_Init and MPI_Finalize functions. In blue color we can see the running time of the program outside the MPI subroutines. First there is the parallel reading of the image, computing g_T^0 coefficients and construction of the initial segmentation function, and then we can see 50 iterations of the RED-BLACK SOR method, indicated by yellow lines corresponding to data exchanges. Figure 17 shows the zoom of the previous visualization at the start of the iteration process. The time spent in the MPI_Allreduce subroutine by every process is visualized in orange. This corresponds to a synchronization of all processes at the first computation of initial residual before starting iterations of the RED-BLACK SOR method. Next, MPI_Allreduce we can see almost on the right end of the picture, when the residual after ten RED and ten BLACK iterations is computed. Figure 18 zooms in on this part of the parallel run. Together with MPI_Allreduce in orange, we can see the MPI_Wait procedure in red, the data exchanges expressed by the yellow lines, and

```
if (myid==0)
{
 for (i=p+1;i<=n1+p-1;i++)
 for (j=p+1;j<=n2+p-1;j++)
 for (k=p+1;k<=n3+p-1;k++)
  {if ((i+j+k) % 2 ==0)
   {z=(b[i][j][k]+aw[i][j][k]*u[i-1][j][k]+
       ae[i][j][k]*u[i+1][j][k]+as[i][j][k]*u[i][j-1][k]+
       an[i][j][k]*u[i][j+1][k]+ab[i][j][k]*u[i][j][k-1]+
       at[i][j][k]*u[i][j][k+1])/ap[i][j][k];
    u[i][j][k]=u[i][j][k]+omega*(z-u[i][j][k]);}}
}
if ((myid>0)&&(myid<nprocs-1))
{
 for (i=p;i<=n1+p-1;i++)
 for (j=p+1;j<=n2+p-1;j++)
 for (k=p+1;k<=n3+p-1;k++)
  {if ((i+j+k) % 2 ==0)
  {z=(b[i][j][k]+aw[i][j][k]*u[i-1][j][k]+
      ae[i][j][k]*u[i+1][j][k]+as[i][j][k]*u[i][j-1][k]+
      an[i][j][k]*u[i][j+1][k]+ab[i][j][k]*u[i][j][k-1]+
      at[i][j][k]*u[i][j][k+1])/ap[i][j][k];
   u[i][j][k]=u[i][j][k]+omega*(z-u[i][j][k]);}}
}
if (myid==nprocs-1)
{
 for (i=p;i<=n1last+p-1;i++)
 for (j=p+1;j<=n2+p-1;j++)
 for (k=p+1;k<=n3+p-1;k++)
  {if ((i+j+k) % 2 ==0)
  {z=(b[i][j][k]+aw[i][j][k]*u[i-1][j][k]+
      ae[i][j][k]*u[i+1][j][k]+as[i][j][k]*u[i][j-1][k]+
      an[i][j][k]*u[i][j+1][k]+ab[i][j][k]*u[i][j][k-1]+
      at[i][j][k]*u[i][j][k+1])/ap[i][j][k];
   u[i][j][k]=u[i][j][k]+omega*(z-u[i][j][k]);}}
}
```

Figure 13. One iteration for RED elements in the parallel RED-BLACK SOR method. One iteration for BLACK elements differs only in the Boolean condition of the **if** command, which has the form $(i + j + k)\ \%\ 2\ ==\ 1$.

of course in blue we can see the running of the program in updating the values of $u[i][j][k]$, either for all RED elements or for all BLACK elements by every process. Figure 19 gives insight into the MPI data transmission process. The time spent in MPI_Isend is shown in violet, MPI_Irecv in gray, and we can see again MPI_Wait in red, which controls so that the desired data are completely transfered from one process to another.

```
if (myid==0)
{
 MPI_Isend(&(u[n1][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid+1,7,MPI_COMM_WORLD,&req1);
 MPI_Irecv(&(u[n1+p][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid+1,7,MPI_COMM_WORLD,&req2);
}
if ((myid>0)&&(myid<nprocs-1))
{
 MPI_Isend(&(u[n1][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid+1,7,MPI_COMM_WORLD,&req1);
 MPI_Isend(&(u[p][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid-1,7,MPI_COMM_WORLD,&req2);
 MPI_Irecv(&(u[n1+p][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid+1,7,MPI_COMM_WORLD,&req3);
 MPI_Irecv(&(u[0][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid-1,7,MPI_COMM_WORLD,&req4);
}
if (myid==nprocs-1)
{
 MPI_Isend(&(u[p][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid-1,7,MPI_COMM_WORLD,&req1);
 MPI_Irecv(&(u[0][p+1][p+1]),(n2-1)*(n3-1),MPI_DOUBLE,
           myid-1,7,MPI_COMM_WORLD,&req2);
}
MPI_Wait(&req1, &status);
MPI_Wait(&req2, &status);
if (myid > 0 && myid < nprocs-1)
{
 MPI_Wait(&req3, &status);
 MPI_Wait(&req4, &status);
}
```

Figure 14. Point-to-point communication after one RED-BLACK SOR iteration for RED elements. The same exchange of data is done also after one iteration for BLACK elements.

5. DISCUSSION OF COMPUTATIONAL RESULTS

In this section we discuss the numerical examples computed using scheme (18) and by its parallel implementation as presented in the previous section.

In [56] we have shown, using comparisons of our numerical solutions with known nontrivial exact solutions, that the 3D semi-implicit co-volume method (18) is second-order accurate for smooth (or mildly singular) solutions and first-order accurate for highly singular solutions (when the gradient is vanishing on a large subset of a domain and a discontinuity set of the gradient field is nontrivial). This means that the method is experimentally convergent and reliable for computing graph evolutions forming the flat regions as arising in the subjective surface segmentation method.

```
deltain=0;
if (myid==0)
  {
   for (i=p+1;i<=n1+p-1;i++)
   for (j=p+1;j<=n2+p-1;j++)
   for (k=p+1;k<=n3+p-1;k++)
   {
    deltain+=sqr(-aw[i][j][k]*u[i-1][j][k]-ae[i][j][k]*u[i+1][j][k]
                 -as[i][j][k]*u[i][j-1][k]-an[i][j][k]*u[i][j+1][k]
                 -ab[i][j][k]*u[i][j][k-1]-at[i][j][k]*u[i][j][k+1]
                 +ap[i][j][k]*u[i][j][k]-b[i][j][k]);
   }
  }
if ((myid>0)&&(myid<nprocs-1))
  {
   for (i=p;i<=n1+p-1;i++)
   for (j=p+1;j<=n2+p-1;j++)
   for (k=p+1;k<=n3+p-1;k++)
   {
    deltain+=sqr(-aw[i][j][k]*u[i-1][j][k]-ae[i][j][k]*u[i+1][j][k]
                 -as[i][j][k]*u[i][j-1][k]-an[i][j][k]*u[i][j+1][k]
                 -ab[i][j][k]*u[i][j][k-1]-at[i][j][k]*u[i][j][k+1]
                 +ap[i][j][k]*u[i][j][k]-b[i][j][k]);
   }
  }
if (myid==nprocs-1)
  {
   for (i=p;i<=n1last+p-1;i++)
   for (j=p+1;j<=n2+p-1;j++)
   for (k=p+1;k<=n3+p-1;k++)
   {
    deltain+=sqr(-aw[i][j][k]*u[i-1][j][k]-ae[i][j][k]*u[i+1][j][k]
                 -as[i][j][k]*u[i][j-1][k]-an[i][j][k]*u[i][j+1][k]
                 -ab[i][j][k]*u[i][j][k-1]-at[i][j][k]*u[i][j][k+1]
                 +ap[i][j][k]*u[i][j][k]-b[i][j][k]);
   }
  }
MPI_Allreduce(&deltain,&tmp,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
deltain=tmp;
```

Figure 15. Computing the initial residual in parallel.

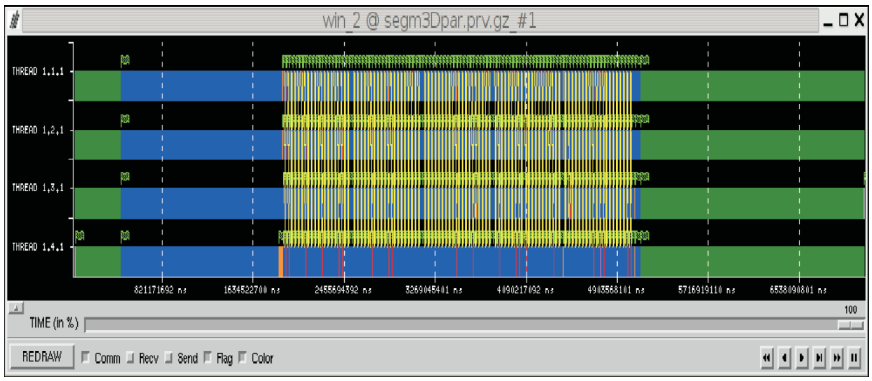

Figure 16. Visualization of the parallel run on 4 processors. See attached CD for color version.

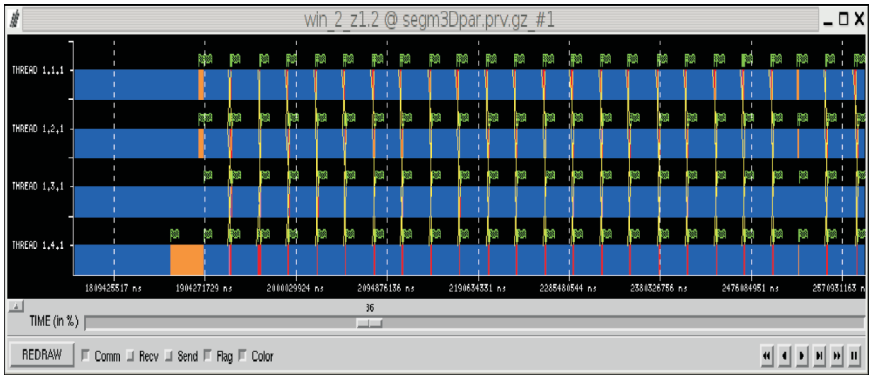

Figure 17. Zoom of the parallel run. See attached CD for color version.

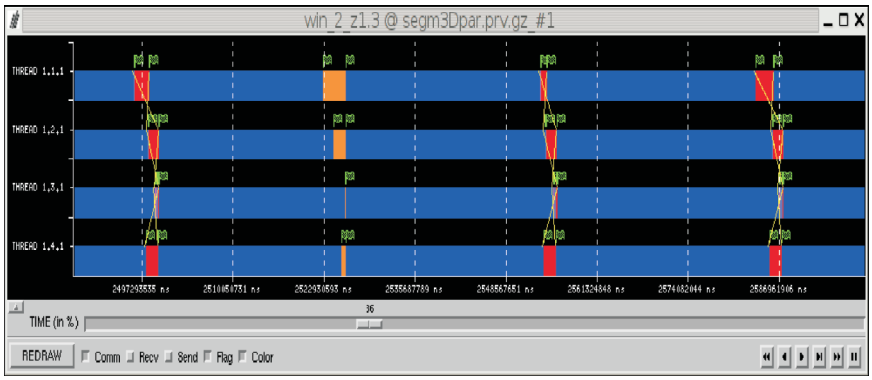

Figure 18. Further zoom of the parallel run. See attached CD for color version.

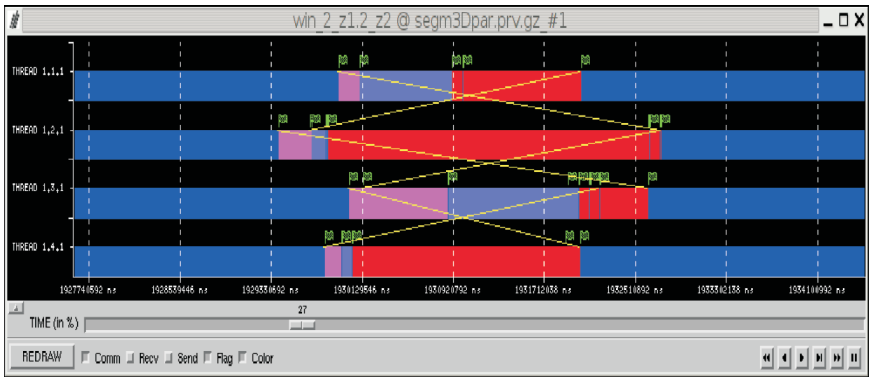

Figure 19. Final zoom of the parallel run. See attached CD for color version.

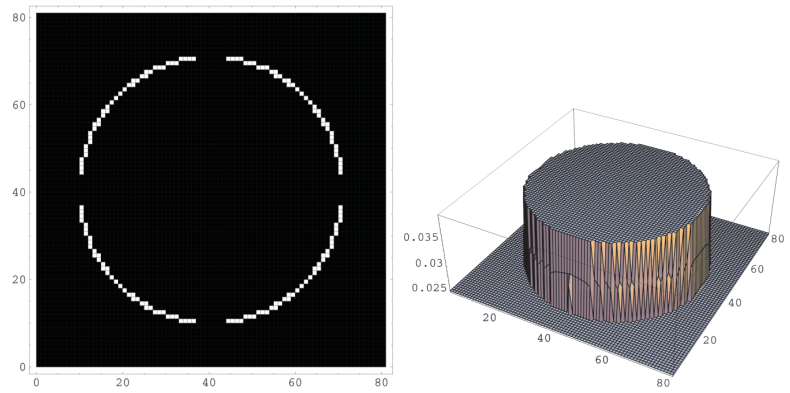

Figure 20. Subjective surface segmentation of 3D sphere with four holes. See attached CD for color version.

As a first example, we segment a 3D image with 81^3 voxels containing a white sphere with four holes on a black background. One 2D slice along the equator is presented in Figure 20 (left). On the right-hand side of Figure 20 we can see a 2D cut of the segmentation function in the same 2D slice after 28 steps of the semi-implicit scheme using $\tau = 0.002$, $\varepsilon = 10^{-3}$, $K = 1$, $TOL = 0.001$ and $\delta = 10^{-5}$. This state of the segmentation function with a shock profile along the edges continuing also into gaps can be successively used to segment the sphere with completion of the holes. The result, where we visualize level surface 0.03, is plotted in Figure 21.

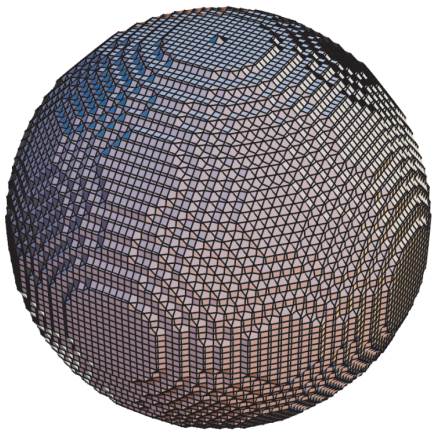

Figure 21. Result of subjective surface segmentation of a 3D sphere with four holes. See attached CD for color version.

Table 1. Computing times and speed-up of parallel program running on 2 to 32 processors.

# processors	2	4	8	16	32
time(secs)	373.4	198.15	112.5	62.85	38.5
speed-up	2	3.77	6.64	11.88	19.4
rel. speed-up	1	0.94	0.83	0.74	0.61

In the next example, we solve the same problem but with an image resolution given by 128^3 voxels. This 3D example we solved on an MPP cluster at CINECA in Bologna. Since due to the huge amount of unknowns we cannot solve the problem on a single processor, we started the report on the results with computation on two processors. Table 1 shows the computing times in seconds for 34 time steps when the segmentation was achieved using the same parameters as above. As we can see, the computation times are well scaled using a larger number of processors. As expected, due to the increasing complexity of communication using a large number of processors, the relative speed-up (i.e., speed-up over a number of processors) is decreasing.

Next, we present an example of subjective surface segmentation of a 3D echocardiographic image of size $81 \times 87 \times 166$ voxels. We use $\tau = 0.001$, $K = 1, TOL = 0.001$, and $\delta = 10^{-5}$. As one can see from the volume rendering visualization in Figure 22, the 3D image is very noisy; however, the surface of the left ventricle is observable. How noisy is the image intensity can be seen also from Figure 23, where we plot intensity and its graph in one 2D slice. Due to the high complexity of this image, we start the segmentation process with an initial function with maxima in several "points of view" inside the desired object. We again evolve the segmentation function until the L_2 norm of the difference of the two subsequent time steps is less than the prescribed threshold δ. We then check a 2D slice with relatively good ventricular boundary edges (Figure 24), where we can see an accumulation of level sets along the inner boundary of the ventricle (Figure 24, left). The largest gap in the histogram (Figure 24, right) indicates the shock in the segmentation function, which can be used for segmentation. We choose one level inside the gap, and plot it inside the slice (Figure 25, left). We can check what this level set looks like in other noisy slices (Figure 25, right, Figure 27), and then we visualize the corresponding 3D isosurface (Figure 28), which gives a realistic representation of the left ventricle.

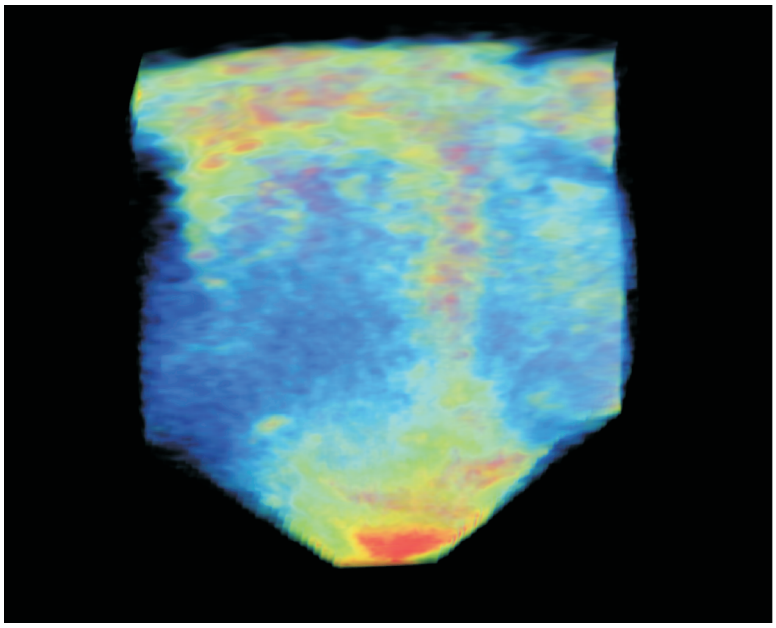

Figure 22. Volume rendering of the original 3D data set. See attached CD for color version.

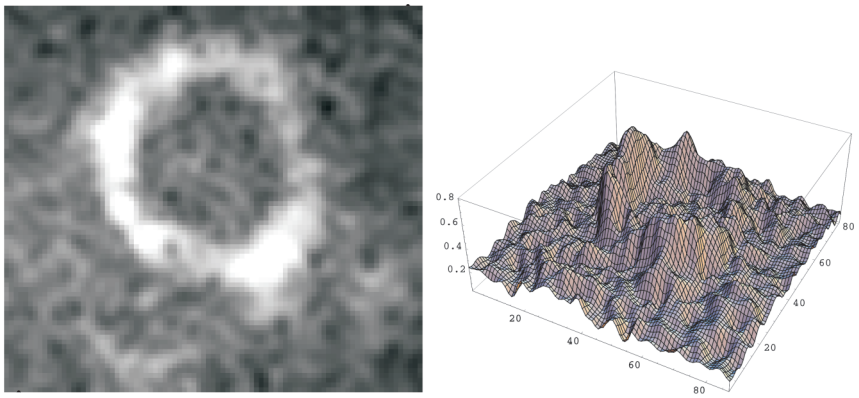

Figure 23. Plot of image intensity in slice $k = 130$ (left), and its 3D graphical view (right). See attached CD for color version.

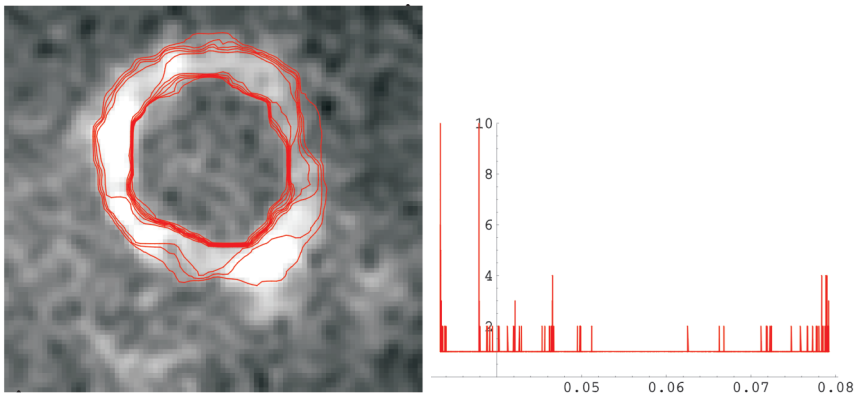

Figure 24. Plot of accumulated level sets in slice $k = 130$ (left); the histogram of the segmentation function in this slice (right). See attached CD for color version.

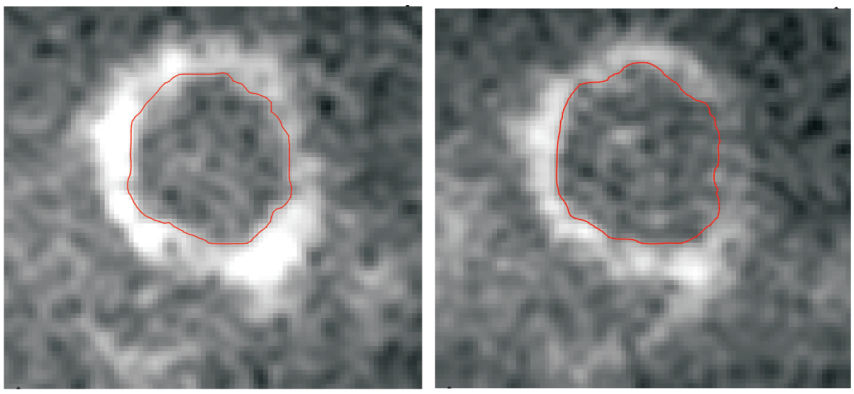

Figure 25. Plot of image intensity together with level line 0.052 in slices $k = 130$ (left) and $k = 125$ (right). Visualization of 3D surface in Figure 22 is done with the same level set. See attached CD for color version.

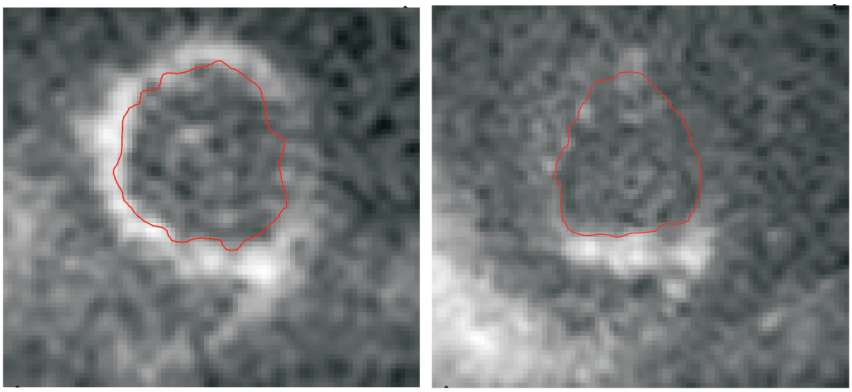

Figure 26. Plot of image intensity together with level line 0.052 in slices $k = 115$ (left) and $k = 100$ (right). Visualization of 3D surface in Figure 22 is done with the same level set. See attached CD for color version.

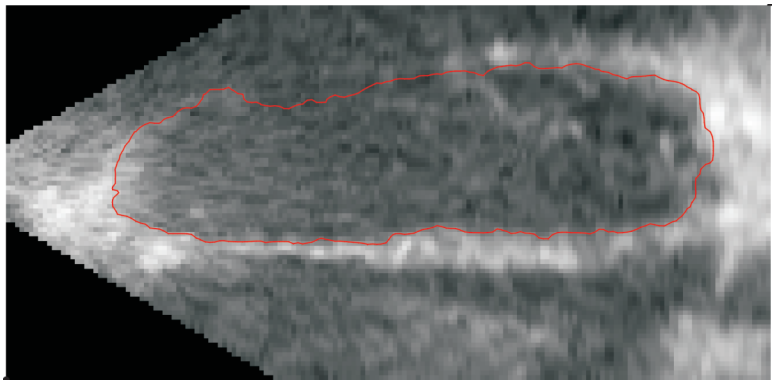

Figure 27. Plot of image intensity together with level line 0.052 in two other slices, $j = 40$. Visualization of 3D surface in Figure 22 is done with the same level set. See attached CD for color version.

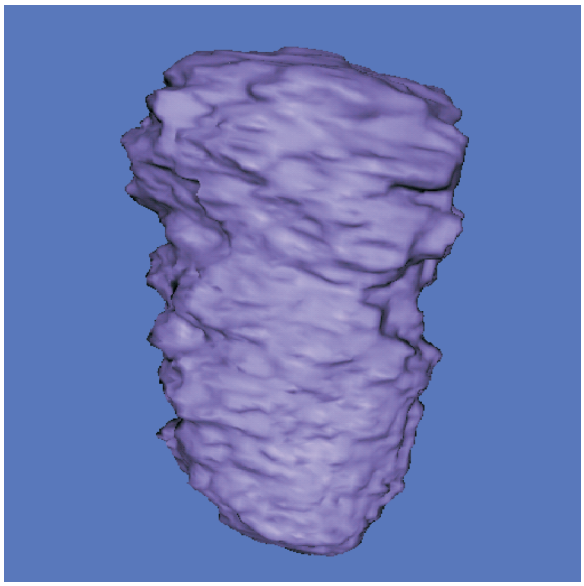

Figure 28. Isosurface visualization of the segmentation result for the left ventricle. See attached CD for color version.

6. ACKNOWLEDGMENTS

This work was supported by Project HPC-EUROPA at the CINECA Super-Computing Center, Bologna, by EU projects Embryomics, and BioEmergences and by grants VEGA 1/3321/06 and APVT-20-040902. We thank to G. Ballabio, C. Calonaci, and R. Gori from CINECA for an introduction to MPI parallel programming and for 3D visualizations.

7. REFERENCES

1. Kass M, Witkin A, Terzopulos D. 1987. Snakes: active contour models. *Int J Comput Vision* **1**:321–331.
2. Gage M, Hamilton RS. 1986. The heat equation shrinking convex plane curves. *J Diff Geom* **23**:69–96.
3. Grayson M. 1987. The heat equation shrinks embedded plane curves to round points. *J Diff Geom* **26**:285–314.
4. Dziuk G. 1991. Algorithm for evolutionary surfaces. *Numer Math* **58**:603–611.
5. Dziuk G. 1994. Convergence of a semi-discrete scheme for the curve shortening flow. *Math Models Methods Appl Sci* **4**:589–606.
6. Dziuk G. 1999. Discrete anisotropic curve shortening flow. *SIAM J Numer Anal* **36**:1808–1830.

7. Mikula K, Ševčovič D. 2001. Evolution of plane curves driven by a nonlinear function of curvature and anisotropy. *SIAM J Numer Anal* **61**:1473–1501.
8. Mikula K, Ševčovič D. 2004. Computational and qualitative aspects of evolution of curves driven by curvature and external force. *Comput Visualiz Sci*, **6**(4):211–225.
9. Mikula K, Ševčovič D. 2004. A direct method for solving an anisotropic mean curvature flow of planar curve with an external force. *Math Methods Appl Sci* **27**(13):1545–1565.
10. K. Mikula, Ševčovič D. 2006. Evolution of curves on a surface driven by a geodesic curvature and external force. *Applic Anal* **85**(4):345–362.
11. Osher S, Sethian JA. 1988. Front propagating with curvature dependent speed: algorithms based on the Hamilton-Jacobi formulation. *J Comput Phys* **79**:12–49.
12. Sethian JA. 1990. Numerical algorithm for propagating interfaces: Hamilton–Jacobi equations and conservation laws. *J Diff Geom* **31**:131–161.
13. Sethian JA. 1999. Level set methods and fast marching methods. In *Evolving interfaces in computational geometry, fluid mechanics, computer vision, and material science*. Cambridge: Cambridge UP.
14. Osher S, Fedkiw R. 2003. *Level set methods and dynamic implicit surfaces*. New York: Springer.
15. Sapiro G. 2001. *Geometric partial differential equations and image analysis*. Cambridge: Cambridge UP.
16. Handlovičová A, Mikula K, Sgallari F. 2003. Semi-implicit complementary volume scheme for solving level set-like equations in image processing and curve evolution. *Numer Math* **93**:675–695.
17. Frolkovič P, Mikula K. 2003. *Flux-based level set method: a finite volume method for evolving interfaces*. Preprint IWR/SFB 2003-15, Interdisciplinary Center for Scientific Computing, University of Heidelberg.
18. Frolkovič P, Mikula K. 2005. *High resolution flux-based level set method*. Preprint 2005-12, Department of Mathematics and Descriptive Geometry, Slovak University of Technology, Bratislava.
19. Caselles V, Catté F, T. Coll, Dibos F. 1993. A geometric model for active contours in image processing. *Numer Math* **66**:1–31.
20. Malladi R, Sethian JA, Vemuri B. 1995. Shape modeling with front propagation: a level set approach. *IEEE Trans Pattern Anal Machine Intell* **17**:158–174.
21. Perona P, Malik J. 1990. Scale space and edge detection using anisotropic diffusion. *IEEE Trans Pattern Anal Machine Intell* **12**(7):629–639.
22. Catté F, Lions PL, Morel JM, Coll T. 1992. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM J Numer Anal*, **29**:182–193.
23. Weickert J, Romeny BMtH, Viergever MA. 1998. Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Trans Image Processing* **7**:398–410.
24. Kačur J, Mikula K. 1995. Solution of nonlinear diffusion in image smoothing and edge detection. *Appl Numer Math* **17**:47–59.
25. Kačur J, Mikula K. 2001. Slow and fast diffusion effects in image processing. *Comput Visualiz Sci* **3**(4):185–195.
26. Mikula K, Ramarosy N. 2001. Semi-implicit finite volume scheme for solving nonlinear diffusion equations in image processing. *Numer Math* **89**(3):561–590.
27. Mikula K, Sgallari F. 2003. Semi-implicit finite volume scheme for image processing in 3D cylindrical geometry. *J Comput Appl Math* **161**(1):119–132.
28. Mikula K. 2002. Image processing with partial diferential equations. In *Modern methods in scientific computing and applications*, pp. 283–321. Eds A Bourlioux, MJ Gander. NATO Science Ser. II, Vol. 75. Dodrecht: Kluwer Academic.
29. Krivá Z, Mikula K. 2002. An adaptive finite volume scheme for solving nonlinear diffusion equations in image processing. *J Vis Commun Image Represent* **13**:22–35.
30. E. Bänsch, Mikula K. 1997. A coarsening finite element strategy in image selective smoothing. *Comput Visualiz Sci* **1**(1):53–61.

31. E. Bänsch, Mikula K. 2001. Adaptivity in 3D image processing. *Comput Visualiz Sci* **4**(1):21–30.
32. Sarti A, Mikula K, Sgallari F. 1999. Nonlinear multiscale analysis of three-dimensional echocardiographic sequences. *IEEE Trans Med Imaging* **18**:453–466.
33. Sarti A, Mikula K, Sgallari F, Lamberti C. 2002. Nonlinear multiscale analysis models for filtering of 3D + time biomedical images. In *Geometric methods in biomedical image processing*, pp. 107–128. Ed R Malladi. New York: Springer.
34. Sarti A, Mikula K, Sgallari F, Lamberti C. 2002. Evolutionary partial differential equations for biomedical image processing. *J Biomed Inform* **35**:77–91.
35. Alvarez L, Lions PL, Morel JM. 1992. Image selective smoothing and edge detection by nonlinear diffusion, II. *SIAM J Numer Anal* **29**:845–866.
36. L Alvarez, Guichard F, Lions PL, Morel JM. 1993. Axioms and fundamental equations of image processing. *Arch Rat Mech Anal* **123**:200–257.
37. Mikula K, Sarti A, Lamberti C. 1997. Geometrical diffusion in 3D-echocardiography. In *Proceedings of ALGORITMY'97, a conference on scientific computing*, pp. 167–181. http://www.math.sk/mikula/msl_alg97.pdf
38. Handlovičová A, Mikula K, Sarti A. 1999. Numerical solution of parabolic equations related to level set formulation of mean curvature flow. *Comput Visualiz Sci* **1.**(2):179–182.
39. Handlovičová A, Mikula K, Sgallari F. 2002. Variational numerical methods for solving nonlinear diffusion equations arising in image processing. *J Vis Commun Image Represent* **13**:217–237.
40. Mikula K. 2001. Solution and applications of the curvature driven evolution of curves and surfaces. In *Numerical methods for viscosity solutions and applications*, pp. 173–196. Eds M Falcone, Ch Makridakis. Advances in Mathematics for Applied Sciences, Vol. 59. Singapore: World Scientific.
41. Mikula K, Preusser T, Rumpf M, Sgallari F. 2002. On anisotropic geometric diffusion in 3D image processing and image sequence analysis. In *Trends in nonlinear analysis*, pp. 307–322. Ed M Kirkilionis, et al. New York: Springer.
42. Mikula K, Preusser T, Rumpf M. 2004. Morphological image sequence analysis. *Comput Visualiz Sci* **6**(4):197–209.
43. Caselles V, Kimmel R, Sapiro G. 1995. Geodesic active contours. In *Proceedings of the fifth international conference on computer vision (ICCV'95)*, pp. 694–699. Washington, DC: IEEE Computer Society.
44. Caselles V, Kimmel R, Sapiro G. 1997. Geodesic active contours. *Int J Comput Vision* **22**:61–79.
45. Caselles V, Kimmel R, Sapiro G, Sbert C. 1997. Minimal surfaces: a geometric three dimensional segmentation approach. *Numer Math* **77**:423–451.
46. Kichenassamy S, Kumar A, Olver P, Tannenbaum A, Yezzi A. 1995. Gradient flows and geometric active contours models. In *Proceedings of the fifth international conference on computer vision (ICCV'95)*, pp. 810–815. Washington, DC: IEEE Computer Society.
47. Kichenassamy S, Kumar A, Olver P, Tannenbaum A, Yezzi A. 1996. Conformal curvature flows: from phase transitions to active vision. *Arch Rat Mech Anal* **134**:275–301.
48. Sarti A, Malladi R, Sethian JA. 2000. Subjective surfaces: a method for completing missing boundaries. *Proc Natl Acad Sci USA* **Vol. 12**(97):6258–6263.
49. Sarti A, Citti G. 2001. Subjective surfaces and riemannian mean curvature flow graphs. *Acta Math Univ Comenianae* **70**(1):85–104.
50. Sarti A, Malladi R, Sethian JA. 2002. Subjective surfaces: a geometric model for boundary completion. *Int J Comput Vision* **46**(3):201–221.
51. Evans LC, Spruck J. 1991. Motion of level sets by mean curvature, I. *J Diff Geom* **33**:635–681.
52. Mikula K, Sarti A, Sgallari F. 2005. Semi-implicit co-volume level set method in medical image segmentation. In *Handbook of biomedical image analysis: segmentation and registration models*, pp. 583–626. Ed JS Suri, D Wilson, S Laxminarayan. New York: Springer.

53. Chen Y-G, Giga Y, Goto S. 1991. Uniqueness and existence of viscosity solutions of generalized mean curvature flow equation. *J Diff Geom* **33**:749-786.
54. Crandall MG, Ishii H, Lions PL. 1992. User's guide to viscosity solutions of second order partial differential equations. *Bull Amer Math Soc* **27**:1–67.
55. Citti G, Manfredini M. 2002. Long time behavior of Riemannian mean curvature flow of graphs. *J Math Anal Appl* **273**(2):353–369.
56. Corsaro S, Mikula K, Sarti A, Sgallari F. 2004. *Semi-implicit co-volume method in 3D image segmentation*. Preprint 2004-12, Department of Mathematics and Descriptive Geometry, Slovak University of Technology, Bratislava.
57. Patankar S. 1980. *Numerical heat transfer and fluid flow*. New York: Hemisphere Publishing.
58. Eymard R, Gallouet T, Herbin R. 2000. The finite volume method. In *Handbook for numerical analysis*, Vol. 7, pp. 715–1022. Ed Ph Ciarlet, JL Lions. New York: Elsevier.
59. Le Veque R. 2002. *Finite volume methods for hyperbolic problems*. Cambridge Texts in Applied Mathematics, Cambridge: Cambridge UP.
60. Brenner SC, Scott LR. 2002. *The mathematical theory of the finite element method*. New York: Springer.
61. Thomée V. 1997. *Galerkin finite element methods for parabolic problems*. Berlin: Springer.
62. Deckelnick K, Dziuk G. 1995. Convergence of a finite element method for non-parametric mean curvature flow. *Numer Math* **72**:197-222.
63. Deckelnick K, Dziuk G. 2000. Error estimates for a semi-implicit fully discrete finite element scheme for the mean curvature flow of graphs. *Interfaces Free Bound* **2**(4):341–359.
64. Deckelnick K, Dziuk G. 2003. Numerical approximation of mean curvature flow of graphs and level sets. In *Mathematical aspects of evolving interfaces*, pp. 53–87. Ed L Ambrosio, K Deckelnick, G Dziuk, M Mimura, VA Solonnikov, HM Soner. New York: Springer.
65. Walkington NJ. 1996. Algorithms for computing motion by mean curvature. *SIAM J Numer Anal*, **33**(6):2215–2238.
66. Saad Y. 1996. *Iterative methods for sparse linear systems*. Spanish Fork, UT: PWS Publishing.
67. Aoyama Y, Nakano J. 1999. *RS/6000 SP: Practical MPI Programming, IBM* www.redbooks.ibm.com.
68. Mikula K. 2001. Parallel filtering of three dimensional image sequences. In *Science and supercomputing at CINECA*, pp. 674–677. Ed F Garofalo, M Moretti, M Voli. Bologna: CINECA.
69. Kanizsa G. 1979. *Organization in vision*. New York: Praeger.
70. Mikula K, Sarti A, Sgallari F. 2006. Co-volume method for Riemannian mean curvature flow in subjective surfaces multiscale segmentation. *Comput Visualiz Sci* **9**(1):23–31.