A PARALLEL ALGORITHM FOR SOLVING THE EIKONAL EQUATION

Eva Dejnožková and Petr Dokládal

School of Mines of Paris, Center of Mathematical Morphology, 35, Rue Saint Honoré,
77 300 Fontainebleau, FRANCE, e-mail: {dejnozke,dokladal}@cmm.ensmp.fr

ABSTRACT

A large variety of methods based on Partial Differential Equations (PDE) use the interface propagation. For their flexibility these methods are being more and more applied to various problems ranging from physics, fluid mechanics to control theory and computer vision. The solution of the PDE-based interface evolution is in itself a complex iterative computational task involving a great number of iterations (unknown a priori). Therefore, these applications are very demanding on the hardware and their real-time implementation is still a challenging problem. An efficient implementation could be done by using a specific parallel architecture.

This paper proposes an original, entirely parallel algorithm to solve the Eikonal equation. Which is the base of applications using a weighted distance function. This algorithm alllows the parallel implementation of active contours methods or continuous watershed on a specific hardware.

1. INTRODUCTION

Recently, the image processing methods based on partial differential equations have gotten an ever increasing attention. The application domains include segmentation by active contours, shape from shading, object tracking and shortest path computation. [1]. The formulation of a large number of operators, using the description of interface evolution, leads to the Eikonal equation:

$$|\nabla u| F = 1 \tag{1}$$

In the context of the methods referenced above, the equation (1), has often to be solved repetitively. Furthermore, this calculus represents only a support for other application-specific computation. For this reason, it has to be done efficiently.

Current algorithms used to solve this equation are from the implementation aspect optimal only for a particular type of applications. Two types of algorithms exist: the ones proceed by scanning the entire image [2], the others propagate a narrow-band solution from the source [1], [3]. The algorithms using multiple scannings are simple to implement. However, they have a drawback: the next scan cannot begin before the previous one ends. Therefore, the scanning-based methods are optimal only for those applications where the solution on the entire image is required. Moreover, it is not simple (or even possible) to calculate the influence zones coming from different sources. The algorithms operating in the narrowband are more appropriate for the propagation of labels. On the other hand, they rely in sophisticated data structures. Their implementation is more complex and it is even more difficult to conceive a specific hardware.

The paper proposes a new parallel algorithm called Massive Marching, close by its straightforwardness to the scanning-based algorithms, but derived from methods working in the narrowband with all their advantages. Section 3 presents the principles of Massive Marching, followed by the discussion of the error and the complexity. Section 4 gives some examples of applications of the algorithm.

2. DEFINITIONS

The initial condition to solve Eq. (1) is the curve $\mathcal{C}_0$ placed in a continuous space. This curve is represented implicitly by the (discrete) distance u to $\mathcal{C}_0$ [1]. $\mathcal{C}_0$ is therefore the zero-level set of the function u.

$$\mathcal{C}_0 = \left\{(x,y) \in \mathbb{R}^2 | u(x,y) = 0\right\} \tag{2}$$

Given the curve $\mathcal{C}_0$, we want to find u.

Throughout this paper we use the following notations. Let $p = [x_p, y_p]$ be a point of an isotropic, rectangular and unitary grid. $\mathcal{P}$ denotes the set of all points in an image. $V(p)$ denotes the neighborhood of p defined as $V(p) = \{[x_p,\ y_p \pm 1],\ [x_p \pm 1, y_p]\}$. The point q is a neighbor of p if $q \in V(p)$. $u(p)$ denotes the value of the distance function in p. To obtain u given the initial condition, we assume that the value of distance u in point p is a function of its neighborhood :

$$u(p) = u_{min}(p) + f_{\mathit{diff}}(|u_x(p) - u_y(p)|, \mathcal{F}(p)) \tag{3}$$

where $u_x(p)$, $u_y(p)$ and $u_{min}(p)$ are defined as :

$$u_x(p) = \min\left\{u([x_p+1, y_p]), u([x_p-1, y_p])\right\} \tag{4}$$

$$u_y(p) = \min\left\{u([x_p, y_p+1]), u([x_p, y_p-1])\right\} \tag{5}$$

$$u_{min}(p) = \min\left\{u_x(p), u_y(p)\right\} \tag{6}$$

and $\mathcal{F}(p)$ is some weight. Suppose that f_{diff} is increasing and strictly positive with respect to $|u_x - u_y|$. The formulation of f_{diff} depends on the choice of the numerical scheme.

3. MASSIVE MARCHING

3.1. Algorithm

The algorithm consists in two stages: initialization and propagation. During the initialization stage, the values of the neighbors of $\mathcal{C}_0$ are obtained by some interpolation method. The choice of the interpolation depends on the requirements of the application. One can use either a constant value either a bilinear or a more sophisticated interpolation method allowing to detect more or less complicated forms (for examples see [4], [5]). The second stage consists in propagating the distance from the curve.

Let $\mathcal{A}$ be the set of points initialized by the interpolation. Let $\mathcal{Q}$ be the set of points marked as active $\mathcal{Q} = \{q_i \mid q_i \notin \mathcal{A} \text{ and } V(q_i) \cap \mathcal{A} \neq \emptyset\}$. The algorithm reads as follows:

Initialization

• Initialize the neighborhood of the curve with a signed distance (set $\mathcal{A}$)

• Initialize the distance value u of the other points to ∞

• Mark the neighbors of $\mathcal{A}$ as *active* (set $\mathcal{Q}$)

Propagation

while $\mathcal{Q} \neq \{\}$, do in parallel for all $p \in \mathcal{Q}$:

{

• *Jacobi step:*

$$u^{n+1}(p) = u^n_{min}(p) + \min\left\{ f_{diff}\left(|u^n_x(p) - u^n_y(p)|, \mathcal{F}(p)\right),\ \mathcal{F}(p)\right\} \quad (7)$$

• *Gauss-Seidel step:*

$$u^{n+1}(p) = u^{n+1}_{min}(p) + \min\left\{ f_{diff}\left(|u^{n+1}_x(p) - u^{n+1}_y(p)|, \mathcal{F}(p)\right),\ \mathcal{F}(p)\right\} \quad (8)$$

• *Activation of new points to process:*

⋆ delete p from $\mathcal{Q}$, insert p in $\mathcal{A}$

⋆ if $u(p) < \text{NB}_{\text{width}}$ then for all $q_i,\ q_i \in V(p)$ such that $u^{n+1}(q_i) - u^{n+1}(p) > \varepsilon(q_i)$ insert $q_i \rightarrow \mathcal{Q}$ (9)

}

where NB_{width} is the desired width of the narrow band[1].

At each iteration, the value is calculated for the *active* points. The calculation is done in two steps named after their Markov properties, as introduced in [6]. The first one, the *Jacobi step*, calculates the value of the distance function at t_{n+1} given the values obtained at t_n. The second one, the *Gauss-Seidel step*, recalculates the distance value at t_{n+1} by using the values obtained at t_{n+1}. The two-step computation allows to process in parallel the values of adjacent points (each one depending on the other). (The values of unprocessed points in t_n are automatically carried over to the next iteration and are noted as values at t_{n+1}.)

[1] To obtain u on the entire image let $\text{NB}_{\text{width}} = \infty$

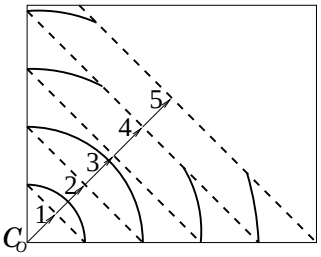


Fig. 1. The front of propagation (dashed lines) and the contours (solid lines) of the distance after five iterations. The source is placed at $\mathcal{C}_0$.

The algorithm does not use any sorted waiting list. Consequently, the front of the propagation is not equidistant to the initialization curve, see Figure 1. Two situations exist where the points that are currently being calculated will have to be reactivated later:

1. The value of the point is calculated on an incomplete neighborhood.
2. The points are activated by a propagation front coming from a source which is not necessarily the closest one.

These two issues are treated by the activation rule.

3.2. Activation rule

The activation rule (9) is derived from the analysis of the numerical scheme. Suppose that $u(p)$ has just been calculated. For a given pixel p, every neighbor verifying (9) is activated in the next iteration while p itself is desactivated. To know whether a neighbor q_i of p should be activated to compute its value $u(q_i)$, we search for an estimator of $u(q_i)$.

From Eq. (3), $f_{diff}(p)$ is the difference between the distance value $u(p)$ of a given point p and the least of the neighbors $u_{min}(p)$. Suppose that p is the least neighbor of q_i. Then q_i receives in the next iteration its value from p. The value $u(q_i)$ will satisfy $u(q_i) \geq u(p) + \inf f_{diff}$.

Suppose $\mathcal{F}(p)$ is an arbitrary but time invariant function then the lower bound $K_{\min}$ of f_{diff} reads

$$K_{\min}(p) = \inf f_{diff}(|u_x(p) - u_y(p)|, \mathcal{F}(p)) = f_{diff}(0, \mathcal{F}(p)) \quad (10)$$

$K_{\min}$ is a predictor of the least increment of u in one iteration according to (9). All neighbors q_i of p such that $u(q_i) - u(p) > K_{\min}(q_i)$ should therefore be (re-)activated and (re-)calculated since the new value $u(p)$ may affect

$u(q_i)$ in the next iteration. Hence, ε must satisfy:

$$\varepsilon(p) \geq K_{\min}(p) > 0 \tag{11}$$

Setting $\varepsilon < K_{\min}$ is useless because it would authorize the activation of points that will not be updated [7]. By setting $\varepsilon > K_{\min}$ one can authorize fewer reactivations (lower execution time) at the price of some error (proportional to ε–$K_{\min}$) in the result.

$K_{\min}$ depends on the numerical scheme. The most often used one is, in the domain of the Level Set, the Godunov scheme [1]:

$$\begin{aligned}&\left[\max\left\{u(p)-u\left([x_p\pm 1,y_p]\right),0\right\}^2-\right.\\&\left.\max\left\{u(p)-u\left([x_p,y_p\pm 1]\right),0\right\}^2\right]^{\frac{1}{2}}=\frac{1}{F(p)}\end{aligned} \tag{12}$$

In order to obtain the maximum values of the terms in Eq. (12) we need to consider the neighbors with minimum values of u and only the neighbors with u lower than $u(p)$. The Godunov scheme requires to determine the maximal solution of a quadratic equation. Then the function f_{diff} in Eq. (3) reads as

$$f_{diff}=\frac{|u_x(p)-u_y(p)|}{2}+\sqrt{\frac{\mathcal{F}(p)^2}{2}-\left(\frac{u_x(p)-u_y(p)}{2}\right)^2} \tag{13}$$

$$K_{\min}(p)=\sqrt{\frac{\mathcal{F}(p)^2}{2}} \tag{14}$$

where $\mathcal{F}(p)=\frac{1}{F(p)}$. From Eq. (1) $\mathcal{F}(p)>0$.

Note that ε is a constant whenever F is constant in (1) and becomes a function of F whenever F varies over the image.

3.3. Estimation of the error

In order to obtain the most accurate solution of (1), all the methods referenced in the introduction allow the points in the image to be recalculated several times. The scanning-based methods recalculate during each scan all the points in the image. Scans have to be repeated unless the convergency. Methods for the narrow band use a variable number of recalculations, implemented by using a sorted heap, depending locally on the neighborhood of every particular point. The number of recalculations is between one and three. At every recalculation the point receives a new value of the distance according to the new values of the neighbors. However, every recalculation does not necessarily result in a lower value. Hence some recalculations are useless. Also Massive Marching authorizes the points to be reactivated and recalculated later.

Massive Marching being a parallel algorithm calculates simultaneously the values of adjacent points, i.e. values depending each on the others. Therefore the calculation is performed in two steps. An additive error at the fourth decimal place may appear in some special cases (as corners etc.), see [7]. The experiments have shown that the two-step calculation gives sufficient accuracy for most practical applications. Should more accurate results be required then the Gauss-Seidel step can be repeated.

The Massive Marching does not suffer from the directional bias induced by the direction of scanning of the neighborhood as Fast Marching.

3.4. Calculation complexity

To estimate the execution time, we outline the calculation complexity of the algorithm.

Upon the initialization, the algorithm calculates by interpolation the values of the points-neighbors of the object(s). The complexity is of $\mathcal{O}(n)$ where n is the number of initialized points.

During the propagation, for every active point the calculation is performed twice, with a constant calculation complexity. Henceafter, the point desactivates itself while activating some of its neighbors. (A point is activated by one of its neighbors whenever the condition (9) is satisfied.) For the sequential implementation of the algorithm, one needs only some FIFO-like data structure to memorize all active points. The complexity of accessing to a FIFO-like structure is constant for both reading and writing.

The overall complexity of Massive Marching is of $\mathcal{O}(n)$ with n being the number of points. Since some points are recalculated because the propagation front is not equidistant to the initial curve C_0, the number of points n can exceed the number of points in the image. The execution time is proportional to the computation complexity.

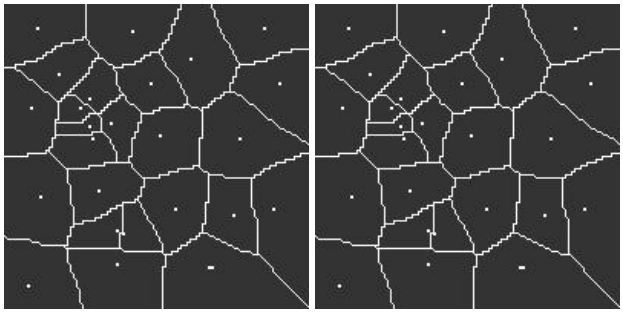
Massive Marching is conceived as parallel. In the case of a fully parallel execution, all operations the currently active points, are performed simultaneously. The execution time is constant for every iteration. The number of iterations is proportional to the maximal distance found in the image.

4. APPLICATIONS

As mentioned in section 1, various types of problems are equivalent to finding the solution of the Eikonal equation (1). In order to prove the validity of the algorithm, several application examples, solved by Massive Marching, are given in this section.

The first example is the computation of the Vorono tesselation for a given set of points in a 2D euclidian space. In this case, we consider $\mathcal{F}(p)=1$ for $\forall p\in\mathcal{P}$ (see Figure 2). Note that if needed, the propagation of labels can be done simultaneously with the propagation of the distance. The result achieved by Massive Marching is compared to the result of Fast Marching [1], which is the one of the most fre-

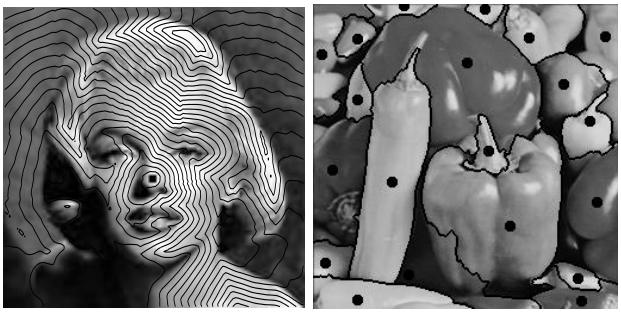
quently used algorithms, based on heap-sorting. The slight difference of the results is due to i)a directional bias of the Fast Marching induced by the direction of the neighborhood scanning and ii)an approximation error of Massive Marching.



(a) Massive Marching (b) Fast Marching

Fig. 2. The Vorono tessellation obtained by Massive Marching, compared to Fast Marching

By letting $\mathcal{F} = I$, where $I(p)$ is a field of strictly positive values, one can obtain a solution which is a weighted distance function. See the contours of a distance weighted by the input image given by Fig. 3(a). This approach is used in such applications as shape from shading or search of the shortest path. If the distance is computed from a given set of markers, the solution is equivalent to a continuous implementation of watershed on the gradient of the input image [8] (see Fig. 3(b)).



(a) (b)

Fig. 3. Applications of weighted distance function: (a) contours of distance weighted by the input image calculated to the source placed in the center of the face, and (b) continuous watershed computed on Sobel gradient of the input image.

5. CONCLUSIONS

This paper proposes an original, fully parallel algorithm to calculate the distance function. The conception of Massive Marching has initially been motivated by the need to accelerate the computation of the distance function in applications of segmentation by active contours. Here, the distance is computed repetitively on a narrow neighborhood of the curve, and therefore needs to be very fast. Massive Marching does not use any sorted waiting lists and allows to obtain the distance in a narrow band.

Its implementation on sequential computers is also very simple. The Massive Marching can be used to compute the distance in a narrow band as well as on the entire image. It supports simultaneous evolution of more curves and the propagation of markers can be done during propagation of the distance. Therefore, it also can yield the watershed or Vorono tessellation.

In such applications where the distance function is smooth and only a few pixels need to be recalculated, the complexity of Massive Marching is close to $\mathcal{O}(n)$. Massive Marching outperforms other algorithms, which are penalized by the access to the sorted heap.

6. REFERENCES

[1] J. Sethian, *Level Set Methods*, Cambridge University Press, 1996.

[2] Y. Tsai, "Rapid and accurate computation of the distance function using grids," Tech. Rep. 17, Department of Mathematics, University of California, Los Angeles, 2000.

[3] P.W. Verbeek and B.J.H. Verwer, "Shading from shape, the eikonal equation solved by grayweighted distance transform," *Pattern Recognition Letters*, vol. 11, no. 10, pp. 681–690, October 1990.

[4] G. Sapiro, *Geometric Partial Differential Equations and Image Analysis*, Cambridge University Press, 2000.

[5] Kaleem Siddiqi, Benjamin B. Kimia, and Chi-Wang Shu, "Geometric shock-capturing ENO schemes for subpixel interpolation, computation and curve evolution," *Graphical models and image processing: GMIP*, vol. 59, no. 5, pp. 278–301, 1997.

[6] M. Boué and P. Dupuis, "Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control," *SIAM Journal on Numerical Analysis*, vol. 36, no. 3, pp. 667–695, 1999.

[7] E. Dejnožková, "Massive marching : A parallel computation of distance function for pde-based applications," Tech. Rep. N-17/02/MM, ENSMP, Center of Mathematical Morphology, 2002.

[8] L. Najman and M. Schmitt, "Watershed of a continuous function," *Signal Processing*, vol. 38, pp. 99–112, July 1994.