MASSIVE MARCHING: A PARALLEL COMPUTATION OF DISTANCE FUNCTION

Eva Dejnozkova Dokládova, Petr Dokládal, Jean-Claude Klein

MASSIVE MARCHING : A PARALLEL COMPUTATION OF DISTANCE FUNCTION

EVA DEJNOŽKOVÁ, PETR DOKLÁDAL AND JEAN-CLAUDE KLEIN

Centre de Morphologie Mathématique, ENSMP
35, Rue Saint Honoré
77 305 Fontainebleau, France
E-mail: dejnozke@cmm.ensmp.fr

The methods based on the evolution of a curve controlled by partial differential equations (PDE), represent an efficient and flexible tool of image segmentation, recognition or object tracking. For these methods, an accurate and rapid computation of the distance function is of a key importance. We propose a new, entirely parallel algorithm yielding the distance function and its hardware implementation.

1. Introduction

The PDE-based methods of image segmentation, object tracking or recognition based on the curve evolution use an implicit description of the curve. This description is realized by a signed distance function to the curve. The curve evolution controlled by the PDE generates a deformation of the distance. Therefore, during the evolution of the curve, the distance function needs to be periodically recalculated.

Below, we discuss the computational difficulties of existing algorithms excluding any efficient hardware implementation. In the Section 3.2 we propose an original parallel algorithm to calculate the distance function to fit the PDE-based application needs, and, in Section 4, its implementation on a specialized architecture.

1.1. *Level Set*

The evolution of the curve $\mathcal{C}$ is represented by the evolution of its implicit description (level set) [5]. According to [1] the signed distance function u (to the curve) is equivalent to the parametric description of the curve and is its unique description. The initial curve is then the zero-level set in u:

$$\mathcal{C}_0 = \big\{(x, y, t) \in \mathbb{R}^2 \,|\, u(x, y, t) = 0\big\} \tag{1}$$

It has been proved that every function u satisfying the Eq. (2) is a signed distance function plus a constant [1].

$$|\nabla u| = 1 \tag{2}$$

In general, the calculation of the distance function comprises three stages: *initialization, approximation* and *propagation*. The *initialization* detects by interpolation the initial position $\mathcal{C}_0$ (Eq. (1)). It is located with a sub-pixel precision. If the linear interpolation is used, its hardware implementation does not rise major problems [7]. Therefore, in the following, we focus only on the approximation and the propagation.

Throughout this paper we use the following notations. Let $p = [x_p, y_p]$ be a point of an isotrope, square unit grid. $V(p)$ 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. The minimum values u of the neighbors of p are:

$$u_x(p) = \min\{u([x_p + 1, y_p]), u([x_p - 1, y_p])\} \tag{3}$$

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

$$u_{min}(p) = \min\{u_x(p), u_y(p)\} \tag{5}$$

2. Approximation: Numerical Scheme

The most often used scheme is, in the domain of the Level Set, the Godunov scheme [5]. This scheme requires to determine the maximum solution of a quadratic equation, repeated for all pairs of neighbors in the directions x and y. Other schemes [3] or [6] e.g., yield u^2. This is extremely unpleasant for applications with subpixel precision, calculated in real numbers, where u is needed (and not u^2).

To decrease the computation complexity, we propose to approximate the distance function in the following way. In general, the value $u(p)$ is obtained as a function of the neighborhood of p as:

$$u(p) = u_{min}(p) + f_{diff}(|u_x(p) - u_y(p)|) \tag{6}$$

where f_{diff} is the function to be approximated. It is an increasing and generally nonlinear function defined on $\langle 0, 1\rangle$ and limited there. As a convenient compromise between the computation complexity and accuracy we have chosen a piecewise linearization in n intervals. The computation complexity is then reduced to the search of the appropriated interval and one addition and one multiplication. Other types of approximations can also be used as look-up-table or addition of a constant (see examples given by Figure 2).

3. Propagation

3.1. *Existing methods*

The *propagation* phase defines the order in which the points receive their values. The Fast Marching [5] is the most often used propagation technique in combination with the PDE-based methods. It computes the distance function by propagating equidistant waves. For this, it uses an *ordered waiting list* with a *real-number priority*. The need of the knowledge of the maximum priority represents a *global information* which makes this algorithm sequential. Moreover, the hardware implementation of waiting lists using a real-number priority is difficult because of such operations like insertion, reading and re-positioning.

Even if other existing methods [6] or [3] e.g., do not use any ordered structure they suffer from other disadvantages as several *obligatory scans of the entire image* in several directions, where the following scan cannot start before the preceding one terminates.

3.2. *Massive Marching*

We propose an original and fully parallel algorithm Massive Marching to calculate the distance function permitting to compute the distance in a narrow band. It does not use any waiting lists. Consequently, the front of the propagation is not equidistant to the initial curve.

The calculation is done in two steps called according to their Markov characteristics [2]. The first one, the *Jacobi step*, calculates the value of the distance function at t_{n+1} given the values calculated at t_n. The second one, the *Gauss-Seidler step*, recalculates the distance value at t_{n+1} by using the values obtained at t_{n+1}. (The values of points not processed in t_n are automatically reported in the next iteration and noted as values at t_{n+1}.)

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

Initialisation
- Initialize the neighborhood of the curve with a signed distance ($\mathcal{A}$)
- Initialize the other points to ∞, the neighbors of $\mathcal{A}$ mark as *active* ($\mathcal{Q}$)

Propagation (unless $\mathcal{Q} \neq \{\}$, for all $p \in \mathcal{Q}$ do in parallel) :
- *Jacobi step:*

$$u^{n+1}(p) = \begin{cases} u^n_{min}(p) + f_{diff}(|u^n_x(p) - u^n_y(p)|) & \text{if} \\ \qquad |u^n_x(p) - u^n_y(p)| \in \langle 0, 1\rangle & \\ u^n_{min}(p) + 1 \quad \text{otherwise} & \end{cases} \tag{7}$$

- *Gauss-Seidler step:*

$$u^{n+1}(p) = \begin{cases} u_{min}^{n+1}(p) + f_{diff}(|u_x^{n+1}(p) - u_y^{n+1}(p)|) & \text{if} \\ \qquad |u_x^{n+1}(p) - u_y^{n+1}(p)| \in \langle 0, 1 \rangle & \\ u^{n+1}(p) = u_{min}^{n+1}(p) + 1 \quad \text{otherwise} & \end{cases} \tag{8}$$

- *Activation of new points to process:*
  - ⋆ delete p from $\mathcal{Q}$, insert p in $\mathcal{A}$
  - ⋆ if $u(p) < \mathrm{NB}_{\mathrm{width}}$ then for all q_i, $q_i \in V(p)$ such that
$$u^{n+1}(q_i) - u^{n+1}(p) > \varepsilon,\ \varepsilon > 0,\ \varepsilon = const. \tag{9}$$
    insert q_i in $\mathcal{Q}$

  where $\mathrm{NB}_{\mathrm{width}}$ is the desired width of the narrow band.

The impact of the two-step based computation on the hardware implementation complexity is negligeable. In fact, the same operation is only computed twice (with entry values from different steps).

The points that may need to be recalculated are detected by using the constant ε (Eq. (9)). Given a pixel p, every neighbor verifying Eq. (9) is activated while p itself is desactivated (see [7] for demonstration of the algorithm). To ensure the increasingness and monotonicity of the distance function, the choice of ε verifies the following:

$$\varepsilon \geq K_{min} > 0 \quad \text{where} \quad K_{min} = \min_{l \in \langle 0,1 \rangle} f_{diff}(l) \tag{10}$$

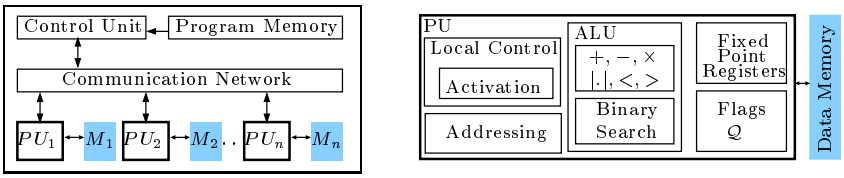
K_{min} is the prediction of the minimum value of the distance increment. By setting $\varepsilon > K_{min}$ we can authorize fewer reactivations (lower execution time) paid by some error in the result (proportional to $\varepsilon - K_{min}$). The Eq. (9) ensures that the algorithm marches forward banning all useless activations.

4. Implementation

The low complexity and straightforwardness of the Massive Marching impose few constraints on the architecture and allow its full parallelisation. It can also be executed in a quasi-parallel way, with more than one point per processing unit.

The global architecture, we propose, is of the divide-and-conquer type with one processing unit per memory block. The block size, realized as a vertical line of the image, is a tradeoff between the processing time and the balanced activity of all the processing units.

The input image is stored in the distributed Data Memory. The user can define the initial curve position and control the direction of its evolution



(a) Global Architecture (b) Architecture of Processing Unit

Figure 1. Implementation of Massive Marching Algorithm

by specifying respectively negative and positive values for the interior and exterior of the curve. The algorithm is read from the Program Memory by the Control Unit and the instructions are sent to the Processing Units. The communication is ensured at two levels: 1) between the Control Unit and the Processing Units (instruction and report of the end of the algorithm), and 2) parallel communication between adjacent Processing Units in the neighborhood.

In order to limit the number of connections between adjacent Processing Units, the values of the east and west neighbors are read in two cycles. In one instruction the unit reads the east neighbor and simultaneously sends its own value to the west neighbor, and inversely in the next instruction.

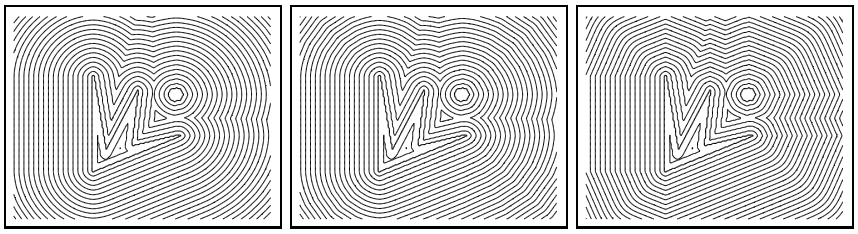
The *initialization* is done by examination of the sign of the neighbors and association of a constant representing the linear interpolation. During the *propagation*, every active point gives birth to one process, executed by one Processing Unit, calculating the distance value from the neighbors.

The block Activation controls the activation of the current point depending on the neighbors and sends the activation command to the neighbors. Also, the Activation block reads the activation flag $\mathcal{Q}$ to provide the instruction *if active then ... else* to prevent useless execution of the given instruction in inactive points.

The value of the current point is computed by a fixed-point ALU. Its complexity depends on the chosen approximation. If the piecewise linearisation is used, the ALU performs a binary search to find the correct values to calculate the piece-wise linearization (see [4]) of the function f_{diff}. The intermediate results and the constants are stored in Fixed Point Registers.

5. Conclusions

We present an original and fully parallel algorithm for calculation of the distance. Contrary to other methods, the Massive Marching completely eliminates the implementation of any priority waiting list and the global



(a) Piece-wise linearisation : 4 intervals

(b) look-up-table, 10 intervales

(c) Addition of a constant, $c = \sqrt{0.5}$

Figure 2. Contours of distance obtained by various approximations; initial curves (placed on the square grid) in heavy lines.

information.

The proposed global architecture of Massive Marching is the divide-and-conquer type with one processing unit per memory block. Each block represents a vertical line of the image in order to reduce the adressing complexity and to balance the activity of all the processing units.

The primary contribution consists in a low-cost, embedded implementation of traditionally costly algorithms based on partial differential equations. The proposed architecture can evolve to integrate also the curve deformation algorithm. Same Processing Units will actually be used to implement the non-linear criteria as energy, curvature or optical flow. The future work will include the implementation of Massive Marching with a chosen application such as segmentation or object tracking.

References

[1] V. I. Arnold. *Geometrical Methods in the Theory of Ordinary Differential Equations*. Springer - Verlag, New York, 1983.

[2] M. Boué and P. Dupuis. Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control. *SIAM Journal on Num. Analysis*, 36:667–695, 1999.

[3] P. Danielsson. Euclidian distance mapping. *Computer graphics and image processing*, 14:227–248, 1980.

[4] T. Gijbels, P. Six, and col. A VLSI architecture for parallel non-linear diffusion with applications in vision. *IEEE, Workshop on VLSI Signal Processing*, 1994.

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

[6] R. Tsai. Rapid and accurate computation of the distance function using grids. Technical Report 00(36), UCLA CAM, 2000.

[7] E. Dejnožková. Massive marching: A Parallel Computation of Distance Function for PDE-based Applications. Technical Report N-17/02/MM, ENSMP, 2002.