Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational Physics 212 (2006) 393–399

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

Short note

O(N) implementation of the fast marching algorithm

Liron Yatziv, Alberto Bartesaghi, Guillermo Sapiro *

Department of Electrical and Computer Engineering, University of Minnesota Minneapolis, 200 Union Street, SE, Minneapolis, MN 55455, USA

Received 2 February 2005; received in revised form 26 July 2005; accepted 5 August 2005
Available online 12 September 2005

Abstract

In this note we present an implementation of the *fast marching algorithm* for solving Eikonal equations that in practice reduces the original run-time from $O(N \log N)$ to *linear*. This lower run-time cost is obtained while keeping an error bound of the same order of magnitude as the original algorithm. This improvement is achieved introducing the straight forward *untidy priority queue*, obtained via a quantization of the priorities in the marching computation. We present the underlying framework, estimations on the error, and examples showing the usefulness of the proposed approach.

Keywords: Fast marching; Hamilton–Jacobi and Eikonal equations; Distance functions; Bucket sort; Untidy priority queue

1. Introduction

The *fast marching method* [15] has been introduced to solve the static Hamilton–Jacobi (Eikonal) equation $|\Delta T|F = 1$, in a computationally efficient way. Here $F > 0$ is the front moving speed and T is the travel time.[1] $1/F$ and T may also be interpreted as travel cost and intrinsic distance respectively. Regarding the computationally efficient implementation of this equation, Tsitsiklis [18] first described an optimal-control approach, while independently Sethian [14] and Helmsen [7] both developed techniques based on upwind numerical schemes. The complexity of their approach is $O(N \log N)$ (Tsitsiklis, [18], also presented $O(N)$

* Corresponding author. Tel.: +1 612 625 1343; fax: +1 612 625 4583.
E-mail addresses: liron@ece.umn.edu (L. Yatziv), abarte@ece.umn.edu (A. Bartesaghi), guille@ece.umn.edu (G. Sapiro).

[1] To avoid abuse of notation, throughout this paper T represents both the analytic solution and the grid values when the equation is later discretized. Our proposed algorithm consistently approximates the solution to the consistent discretization, and then to the continuous Eikonal equation, see below. The mentioned previous algorithms for solving this equation exactly solve the discretized equation.

0021-9991/$ - see front matter
doi:10.1016/j.jcp.2005.08.005

implementations, see below), where N is the total number of grid points.[2] The algorithm is an extension to the classical Dijkstra technique, and is based on finding (at each step) the point with the minimal T value in the *narrow band* set of points that are being updated, and setting it to be *alive* (points that got their final value of T). Neighbors of this point are then updated and the process is repeated until all points in the domain have been processed. All involved numerical computations use upwind schemes. The point with the minimal T value is usually found using a heap priority queue based on a tree. Insertion time to such a heap is $O(\log n)$ where n is the number of points in the narrow band. Due to the positiveness of F, every point is visited at most a constant number of times, and from this the time complexity of the algorithm is $O(N \log N)$. See [15,18] for additional details.

Due to the broad applications of weighted distance functions obtained by solving the above mentioned Eikonal equation, e.g. [8,15], its efficient implementation and numerous extensions have been studied. In particular, fast sweeping algorithms have been proposed, e.g. [2,17,20] (early ideas in this direction were proposed by Danielsson [5]). This technique is based on using a pre-defined sweep strategy, replacing the use of the heap priority queue to find the next point to process, and thereby reducing the overall complexity to $O(N)$.[3]

Here we propose a new implementation of the fast marching algorithm which reduces the computational complexity to $O(N)$ (N being in practice the number of visited points, as in the original fast marching). This is based on the concept that when solving the Eikonal equation for the current grid point, we can quantize (round) the priority values, thereby allowing the use of a table instead of a tree, reducing the updating complexity from $O(\log N)$ to $O(1)$. This is done with the help of a data structure denoted as *untidy priority queue*. We show that the possible error introduced by this simplification can be kept of the same order of magnitude as the numerical error introduced by the spatial discretization inherent to numerical implementations, while in practice, the errors introduced by this approximation are virtually insignificant. In layman words, the idea here proposed is that in the same way that numerical implementations introduce errors due to space discretization, we can also allow for errors in the value computed from the Eikonal equation when used to set the priority for the current grid point. When this is properly done, computational complexity is improved at no cost. The rest of this note provides details on this and examples.

2. Algorithm description

Our method is based on the special properties that grid points hold in the maintained priority queue. When the T value of all points in the queue are always larger or equal than the T value of the latest point extracted from the queue, the queue is denominated as a *monotone priority queue*. This important subclass of priority queues is used in applications such as event scheduling and discrete event simulation. In the fast marching algorithm the queue maintained for the narrow band (NB) has a monotone behavior. If we assume that F is bounded, we can conclude that the T values of the points currently in the queue are also bounded, since there is a maximal possible increment $I_{\max}$ over the point with the minimal T value currently in queue.[4] When a priority queue involves only points with T values in a fixed size range, it is possible to use data structures based on a *circular array*, see Fig. 1. Each entry (bucket) of the circular array contains a list of points with "similar" T value. The *calendar queue* [3] and some of its improvements [1,11,13,16] are all

[2] In practice, the number of visited points during the computation.

[3] Here N is the number of grid points in a bounding box of the region of interest. Although theoretically this is the same "N" as in the original fast marching implementation, it can be much larger in practice (in particular for largely non-uniform speeds F) because fast marching requires computation of visited points only. The constant in the $O(N)$ theoretical complexity of the fast sweeping algorithm is dependent on the speed F, while this dependency doesn't exist in the original fast marching algorithm implementation.

[4] The value of $I_{\max}$ depends on the discretization used. For schemes using canonical $2 \times D$-point neighborhoods, $I_{\max} = \max(1/F)h$, where D is the problem dimension and h the grid size. For schemes using all $3^D - 1$ adjacent neighbors, $I_{\max} = \max(l/F)\sqrt{D}h$.

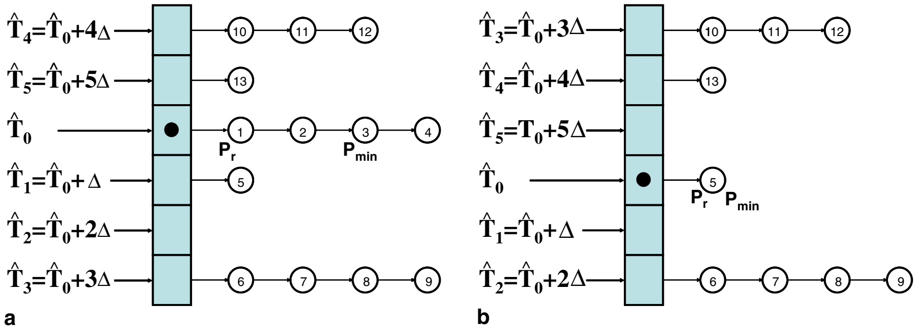

Fig. 1. Operational queue for the narrow band, see text for details.

such queues based on a cyclic bucket sort where the probabilistic complexity of the insert and remove operations is O(1).[5]

Let $P_{\min}$ be the point with time $T_{\min}$ which is the minimal time in the queue and P_r with time T_r be the next point to be removed from the queue. For the calendar queue $P_r = P_{\min}$ always holds, i.e., we are always guaranteed to get the point with the highest priority (lowest T). However, since the fast marching algorithm already has an error dependent on the grid density $h, \mathrm{O}(h)$ for smooth solutions, it is not necessary to strictly keep $P_r = P_{\min}$ to achieve the same general order of numerical error. We propose to use an *untidy priority queue* where $P_r \approx P_{\min}$. The queue is based on the circular array, which is a simplification of the calendar queue. Entries in the circular array of size d represent uniformly quantized levels of T, equally spaced every Δ. If $\widehat{T}$ is the quantized value of T, let $\widehat{T}_0, \widehat{T}_1, \ldots, \widehat{T}_{d-1}$ be the discrete levels of the array entries such that $\widehat{T}_0 < \widehat{T}_1 < \cdots < \widehat{T}_{d-1}$. Entry $\widehat{T}_i$ keeps a FIFO list of grid points with T values in the range $[\widehat{T}_i, \widehat{T}_{i+1}]$ that quantize to $\widehat{T}_i$.

Fig. 1(a) gives an example of the proposed untidy priority queue. In this example $d = 6$ so that $6\Delta = I_{\max}$. Arrival times T are then assumed to be in the range $[\widehat{T}_0, \widehat{T}_0 + I_{\max}]$. The queue keeps track of the memory location L_0 of entry $\widehat{T}_0$ in the array (the entry corresponding to the current value of T). In this example, $\widehat{T}_0$ is the 3rd entry from the top so $L_0 \leftarrow 3$. When a new point P_a with time T_a is added to the queue, according to its quantized time value $\widehat{T}_a$, it is placed in the end of the list of entry T_j, where $j \leftarrow (\widehat{T}_a - \widehat{T}_0)/\Delta$. Finding entry T_j in memory can be done using a simple modulo operation: $L_j = (j + L_0)/\Delta \bmod d$. The next point to be removed from the queue, P_r, will always be the first point in the $\widehat{T}_0$ entry. The figure displays numbers which correspond to the order in which the points would be removed if no new points are added to the queue during the process. Note that due to the quantization, P_r is not necessarily $P_{\min}$, but we can guarantee that $\widehat{T}_{\min} = \widehat{T}_r$ so $0 \leqslant \widehat{T}_r - \widehat{T}_{\min} < \Delta$. When the list at position $\widehat{T}_0$ is emptied, the next non-empty list is used (list at position $\widehat{T}_1$ in this example). The entry $\widehat{T}_0$ is now used to store T values quantized to $\widehat{T}_0 + 6\Delta$ (circular queue). For consistency, all labels are shifted forward one position so the new $P_{\min}$ is now at position $\widehat{T}_0$ as shown in Fig. 1(b).

Note that the quantization is used only to place the grid point in the queue, while the actual T value is used to solve the Eikonal equation when the grid point is selected. Therefore, errors can only occur due to wrong selection order.

The average complexity of the remove operation is O(1) as long as $\mathrm{O}(d) \leqslant \mathrm{O}(n)$, since the operation may involve searching for a non-empty queue (see additional comments at the end of Section 2.1). Selecting a constant size d of order O(1) or by using automatic resizing techniques as presented in [3], it is possible to guarantee a worst case average complexity of O(1). The insert operation has no searching involved and therefore its run-time complexity is O(1).

[5] Yet, worst case complexity is $\mathrm{O}(\log n)$ which happens if too many elements have nearly equal priorities.

An extreme case occurs when $I_{max} \gg 0$ and increments with the order of I_{max} are rare. This causes a "waste" in accuracy, since many discrete levels in the *circular array* would contain empty lists. Prior knowledge of such increment distributions permits to handle the rare large increments separately as suggested in [3], thereby avoiding such a situation.

To conclude, let us point out that Tsitsiklis [18] also described O(N) variations for solving the Eikonal equation (although the constant can be much larger than in his O($N \log N$) approach). His approach, which contains a complete theoretical analysis, is based on buckets and a more elaborated discretization of the equation, which is "more cumbersome and is unlikely to be used when the dimension is higher than three" [18]. His bucket data structure is a linear array of length proportional to max T. In contrast, our approach does not require a new discretization (which in the case of [18] includes an optimization step which is more expensive to solve), uses a cyclic array thereby no needing to estimate the maximal distance T and being more memory efficient, has very small (practical) constant in the O(N) complexity, and quantizes the values only for queue placing and not for actual computation (this causes errors very rarely in practice as detailed below). Thereby, the approach here proposed is simpler, faster, valid for any dimension (as well as for the extensions reported in the literature for computing geodesies on manifolds [9,12]), and memory efficient. In [6] the authors also use a bucket structure in a circular queue to achieve an O(N) implementation of Dijkstra's algorithm, although their work assumes integer increments as in Dial's approach (see also [18] for additional comments and extensions to Dial's algorithm). The authors of [10] also mentioned that numerical precision can be exploited to eliminate the tree search, but no algorithm was proposed.

2.1. Error bounds

Assuming a given state of the algorithm, we would like to bound the additional error in the computation of T of an *alive* point when it is selected out of order using the *untidy priority queue*. Recall that the non-quantized values are always stored in the queue, so an error can result only as a consequence of wrong order in the selection. The analysis presented below corresponds to one single step of the proposed algorithm, and we show below that the error order is the same as in the original fast marching algorithm.

If $P_r \neq P_{min}$ is the next point to be removed from the queue, since the *untidy priority queue* restricts T_r and T_{min} to share the same discrete level, an error $\varepsilon = T_r - T_{min} \leqslant \Delta$ will be incurred. Note that $\varepsilon > 0$ since the algorithm may only increase the value of points in NB, so removing a point early may only cause T_r not to reach T_{min}. Therefore the introduced error can not be negative.

Since $\varepsilon \leqslant \Delta = I_{max}/d$ and $I_{max} = O(h)$, by selecting $d \sim O(l/h)$ we can achieve $\varepsilon = O(h^2)$ (h is the grid size). This is the same error introduced per single step in the original first order fast marching algorithm (for smooth solutions). As in the original algorithm, a finite path has $O(l/h)$ steps,[6] and therefore the accumulated additional error is bounded by O(h). This is the same order of magnitude of error as in the original fast marching algorithm, only that computational complexity is reduced to O(N).[7]

The overall complexity of the whole algorithm is $O([l + d/n]N)$ (the $[1 + d/n]$ term comes from the search for a non-empty bucket as mentioned before plus one access to the element in the non-empty bucket). As long as $O(d) \leqslant O(n)$, $O(1 + d/n) = O(1)$ as stated before, leading to the mentioned complexity of O(N) for the overall algorithm. Since in theory, to preserve the accuracy of the original fast marching algorithm, $d \sim O(l/h)$, it might happen that $O(d) > O(n)$. It is expected that the denser the grid, the larger the average number of points n in the frontier,[8] thereby avoiding this situation of extensively searching for a non-empty

[6] This is due to the use of the FIFO ordering within a bucket, the frontier propagates and "serpentine" type of paths are avoided.

[7] For non-smooth solutions of the Eikonal equation, the fast marching error is larger, and therefore the selection of $d \sim O(l/h)$ is even more favorable.

[8] It is expected that $n \sim O(l/h^{D-1})$, where D is the dimensionality of the grid.

bucket and preserving the overall O(N) complexity (recall also the possibility of using queue resizing techniques as mentioned above).

To conclude let us point out that for more accurate numerical implementations of the fast marching algorithm, the idea here presented of priority quantization can be exploited as well, just using a finer quantization of the order of the desired overall accuracy.

3. Numerical experiments

Numerical experiments show that errors (with respect to the original fast marching algorithm implementation) caused due to the misordering are very rare, small, and in practice far from the upper bounds computed above. In addition to the examples here presented, we tested with the velocity functions in [19] for which the exact solution to the Eikonal equation is known, and find that the algorithm here proposed produced absolutely no additional error compared to the original fast marching algorithm implementation. These examples usually exhibit symmetry and/or regularity in the solution which makes them unsensible to the kind of ordering errors possibly introduced by our algorithm.

Fig. 2 presents error statistics versus the number of discrete levels (errors with respect to the original fast marching algorithm implementation). Numerical experiments show that few discrete levels are enough to achieve a negligible error.

Fig. 3 gives an example of a fast marching application. The task is to segment the lake just by giving 4 points on the shore of the lake, following [4]. The segmentation based on the distance function generated using the *untidy priority queue* is identical to the segmentation archived using an accurate queue. Yet, the run-time when using the *untidy priority queue* is much shorter (see below). Additional examples are available online at http://mountains.ece.umn.edu/~liron/fastmarching/.

Fig. 4 shows the O(N) run-time complexity and execution times for the proposed algorithm. We show that the average number of queue operations done per point in the grid is constant as the grid size N goes to infinity. Using our implementation, the time required to compute distances (from a single seed) in a 2000×2000 grid with random $1/F$ sampled from a uniform distribution in $(0, 1]$, using 1000 discrete buckets, is 1.251 s.

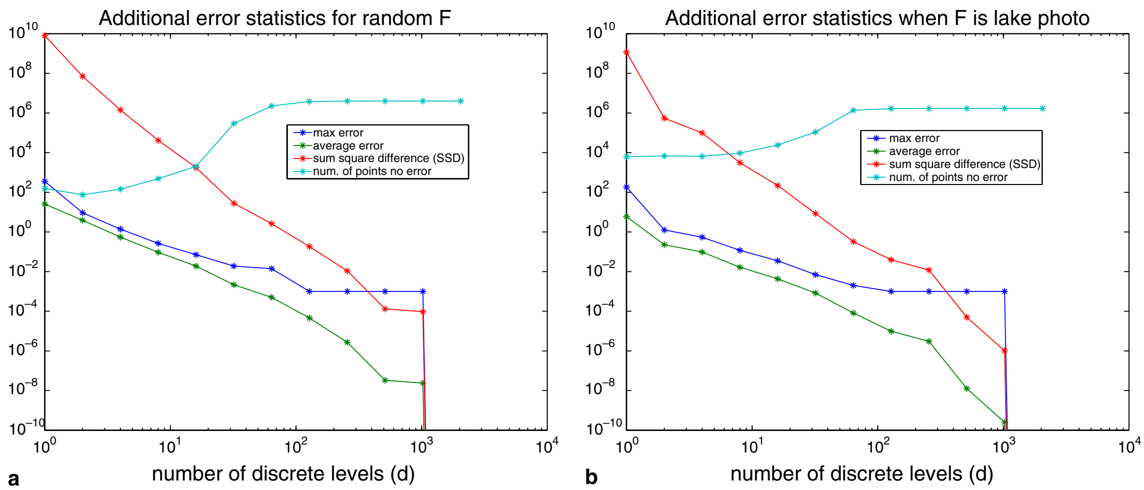

Fig. 2. Error statistics compared to an accurate queue. (a) Statistics when creating a time/distance function on a 2000×2000 grid with random $1/F$ (sampled from a uniform distribution in $(0, 1]$). (b) Same statistics when $1/F$ is a decreasing function of the image gradient, Fig. 3.

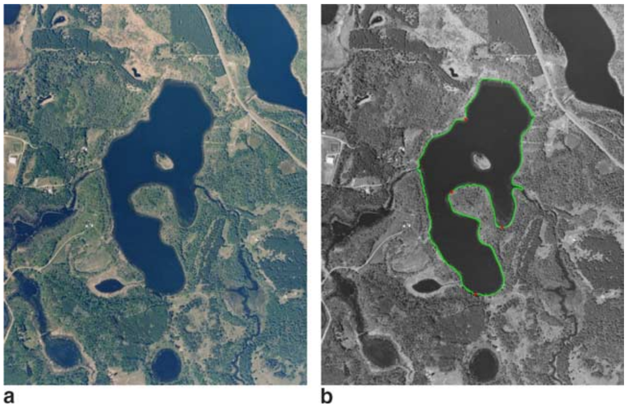

Fig. 3. The segmentation example (note that the lake has been detected, green contour) shows that the quantization error does not interfere with fundamental fast marching applications. (Image size 1163 × 1463). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

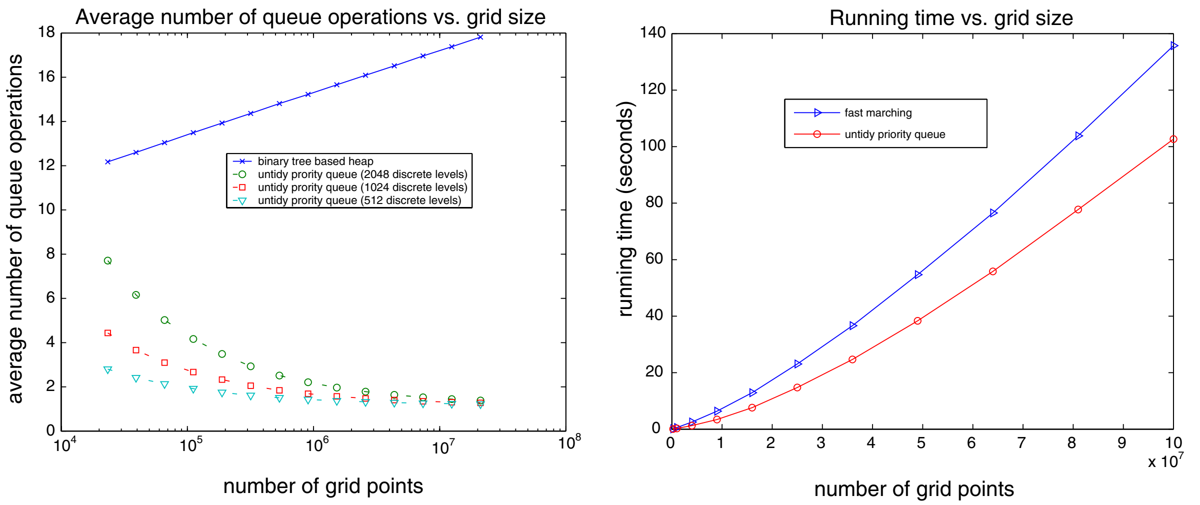

Fig. 4. Comparison of run-time statistics with fast marching on a square grid with random $1/F$. Left: Queue search operations are shown. For the heap operations, these include both the heap access and the priority comparisons. For the untidy priority queue, operations are array entry access and search for a non empty bucket. Right: Overall running times ($d \sim O(1/h)$). Note that we obtain an about constant improvement, regardless of the grid size. This is because the bottleneck is largely determined by factors other than queue management operations alone, e.g., memory allocation. These factors are significantly reduced when, as expected in real applications, computing the distance T is just one part of the whole task (e.g., the data is already in memory).

When F is an edge map obtained from the 1163×1463 image displayed in Fig. 3(a), the time required is 0.591 s. The computer used for the measurements is a Pentium 4 laptop with 2 Ghz processor speed.

4. Concluding remarks

In this note, we have shown a linear complexity implementation of the fast marching algorithm for solving Eikonal equations. The proposed algorithm maintains the overall error and simplicity of the original, higher complexity, implementation. The basic idea is to quantize the priority values and use classical data structures, in particular, an untidy priority queue. The concepts can be applied in any dimension and for solving the Eikonal equation on flat and non-flat domains. To further improve the simple error estimates presented in this work, a detailed study of the probability of misordering due to the priority quantization needs to be pursued. Results in this direction will be reported elsewhere.

Acknowledgments

We thank Gustavo Brown for ideas on queue based implementations, Ron Kimmel for reminding us of the O(N) algorithm in [18], and Laurent Cohen for pointing out the work in [6]. This work was partially supported by the Office of Naval Research, the National Science Foundation, DARPA, the National Institutes of Health, and the National Geospatial-Intelligence Agency.

References

[1] J. Ahn, S. Oh, Dynamic calendar queue, in: Proceedings of the Thirty-Second Annual Simulation Symposium, 1999, p. 20.
[2] M. Boue, P. Dupuis, Markov chain approximation for deterministic control problems with affine dynamics and quadratic cost in the control, SIAM Journal on Numerical Analysis 36 (1999) 667–695.
[3] R. Brown, Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem, Communications of the ACM 31 (1988) 1220–1227.
[4] L.D. Cohen, R. Kimmel, Global minimum for active contours models: A minimal path approach, International Journal of Computer Vision 24 (1997) 57–78.
[5] P. Danielsson, Euclidean distance mapping, Computer Graphics and Image Processing 14 (1980) 227–248.
[6] A.X. Falco, J.K. Udupa, F.K. Miyazawa, An ultra-fast user-steered image segmentation paradigm: Livewire on the fly, IEEE Transactions on Medical Imaging 19 (1) (2000) 55–62.
[7] J. Helmsen, E.G. Puckett, P. Collela, M. Dorr, Two new methods for simulating photolithography development in 3D, in: Proceedings of SPIE Microlithography IX, 1996, p. 253.
[8] R. Kimmel, Numerical Geometry of Images: Theory, Algorithms, and Applications, Springer-Verlag, New York, 2004.
[9] R. Kimmel, J.A. Sethian, Computing geodesic paths on manifolds, Proceedings of the National Academy of Sciences 95 (15) (1998) 8431–8435.
[10] R. Kimmel, J.A. Sethian, Optimal algorithm for shape from shading and path planning, Journal of Mathematical Imaging and Vision 14 (3) (2001) 237–244.
[11] K. Leong Tan, L. Thng, Snoopy calendar queue, in: Proceedings of the 32nd Conference on Winter Simulation, 2000, pp. 487–495.
[12] F. Memoli, G. Sapiro, Fast computation of weighted distance functions and geodesies on implicit hyper-surfaces, Journal of Computational Physics 173 (2) (2001) 730–764.
[13] R. Rönngren, J. Riboe, R. Ayani, Lazy queue: A new approach to implementing the pending-event set, in: Proceedings of the 24th Annual Symposium on Simulation, 1991, pp. 194–204.
[14] J.A. Sethian, A fast marching level-set method for monotonically advancing fronts, Proceedings of the National Academy of Sciences 93 (4) (1996) 1591–1595.
[15] J.A. Sethian, Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Sciences, Cambridge University Press, 1996.
[16] M. Thorup, On RAM priority queues, in: Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms, 1996, pp. 59–67.
[17] Y.R. Tsai, L.T. Cheng, S. Osher, H.K. Zhao, Fast sweeping algorithms for a class of Hamilton–Jacobi equations, SIAM Journal on Numerical Analysis 41 (2) (2002) 673–694.
[18] J.N. Tsitsiklis, Efficient algorithms for globally optimal trajectories, IEEE Transactions on Automatic Control 40 (1995) 1528–1538.
[19] Y. Zhang, J. Qian, H.K. Zhao, High order fast sweeping methods for static Hamilton–Jacobi equations, Journal of Scientific Computing (to appear).
[20] H.K. Zhao, Fast sweeping method for Eikonal equations, Mathematics of Computation 74 (2004) 603–627.