# $O(N)$ IMPLEMENTATION OF THE FAST MARCHING ALGORITHM

By

**Liron Yatziv**

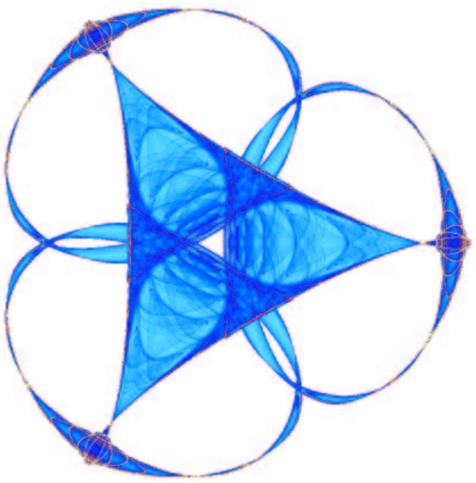
**Alberto Bartesaghi**

and

**Guillermo Sapiro**

**IMA Preprint Series # 2021**

( February 2005 )



# O(N) Implementation of the Fast Marching Algorithm

Liron Yatziv, Alberto Bartesaghi, and Guillermo Sapiro

*Department of Electrical and Computer Engineering, University of Minnesota Minneapolis, MN 55455, USA, {liron,abarte,guille}@ece.umn.edu*

**Abstract**

In this note we present an implementation of the *fast marching algorithm* for solving Eikonal equations that 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.

*Key words:* Fast marching, Hamilton Jacobi and Eikonal equations, distance functions, untidy priority queue.

## 1 Introduction

The *fast marching method* [13] has been introduced to solve the static Hamilton-Jacobi (Eikonal) equation $|\nabla T|F = 1$, in a computationally efficient way. Here $F > 0$ is the front moving speed and $T$ is the travel time. $F$ and $T$ may also be interpreted as travel cost and intrinsic distance respectively. Regarding the computationally efficient implementation of this equation, Tsitsiklis [16] first described an optimal-control approach, while independently Sethian [12] and Helmsen [5] both developed techniques based on upwind numerical schemes. The complexity of their approach is $O(N \log N)$, where $N$ is the total number of grid points. [1] The algorithm is an extension to the classical Dijkstra technique, and is based on finding (at each iteration) 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 already got their definitive $T$ value). Neighbors of this point are then updated and the process is repeated until all points in the

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

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 [13] for additional details.

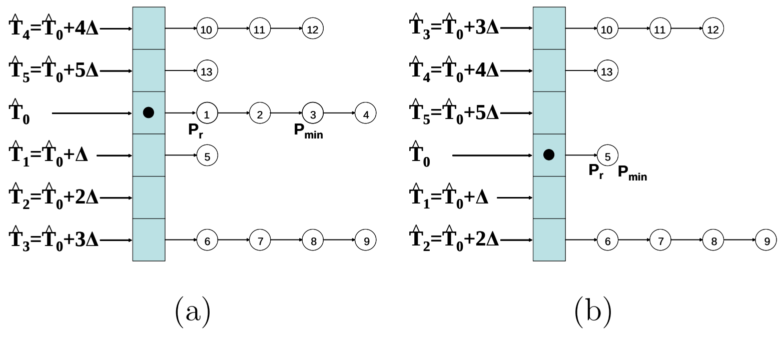
Due to the broad applications of weighted distance functions obtained by solving the above mentioned Eikonal equation, e.g., [6,13], its efficient implementation and numerous extensions have been studied. In particular, fast sweeping algorithms have been proposed, e.g., [2,15,17] (early ideas in this direction were proposed by Danielsson). 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 $O(N)$. [2]

Here we propose a new implementation of the fast marching algorithm which reduces the computational complexity to $O(N)$ (being in practice $N$ as in the original fast marching, the number of visited points). 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 to use 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 theoretical or practical 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

[2] 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 anisotropic speeds $F$ (computations are done only at visited points in fast marching).



$\hat{T}_4=\hat{T}_0+4\Delta$
$\hat{T}_5=\hat{T}_0+5\Delta$
$\hat{T}_0$
$\hat{T}_1=\hat{T}_0+\Delta$
$\hat{T}_2=\hat{T}_0+2\Delta$
$\hat{T}_3=\hat{T}_0+3\Delta$
$P_r$
$P_{min}$
(a)
$\hat{T}_3=\hat{T}_0+3\Delta$
$\hat{T}_4=\hat{T}_0+4\Delta$
$\hat{T}_5=\hat{T}_0+5\Delta$
$\hat{T}_0$
$\hat{T}_1=\hat{T}_0+\Delta$
$\hat{T}_2=\hat{T}_0+2\Delta$
$P_r$ $P_{min}$
(b)

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

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 derive that the $T$ values of the points in the queue are also bounded since there is a maximal possible increment $I_{max}$ over the point with the minimal $T$ value in queue.[3] 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 Figure 1). Each entry of the circular array contains a list of points with "similar" $T$ value. The *calender queue* [3] and some of it's improvements [1,9,11] are all such queues where the probabilistic complexity of the insert and remove operations are $O(1)$.[4]

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 lowest priority. However, since the fast marching algorithm already has an error dependent on the grid density $h$, $O(h)$, 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 calender queue. Each entry in the circular array of size $d$ represents a discrete (quantized) level of $T$, uniformly distributed. Let $\hat{T}$ be the quantized value of $T$ and $\Delta$ be the difference between any two consecutive discrete levels. Let $\hat{T}_0, \hat{T}_1, \ldots, \hat{T}_{d-1}$ be the discrete levels of the array entries such that $\hat{T}_0 < \hat{T}_1 < \ldots < \hat{T}_{d-1}$. The entry $\hat{T}_i$ keeps a FIFO list of grid points with $T$ values in the range $[\hat{T}_i, \hat{T}_{i+1}]$ that quantize to $\hat{T}_i$.

Figure 1(a) gives an example of the proposed untidy priority queue. In this

[3] The value of $I_{max}$ depends on the neighborhood scheme used, *e.g.* for 4-neighbors, $I_{max} = max\ (F)$.

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

example $d = 6$ so that $6\Delta = I_{max}$. Priority values $T$ are then assumed to be in the range $[\hat{T}_0, \hat{T}_0 + I_{max}]$. The queue keeps track of the memory location $L_0$ of entry $T_0$ in the array (the entry corresponding to the current value of $T$). In this example, $T_0$ 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 it's quantized time value $\hat{T}_a$, it is placed in the end of the list of entry $T_j$. Where $j \leftarrow (\hat{T}_a - \hat{T}_0)/\Delta$. Finding entry $T_j$ in memory can be done using a simple modulo operation: $L_j = (j + L_0)/\Delta \ mod \ d$. The next point to be removed from the queue, $P_r$, will always be the first point in the $\hat{T}_0$ 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 $\hat{T}_{min} = \hat{T}_r$ so $0 \leq \hat{T}_r - \hat{T}_{min} < \Delta$. When the list at position $\hat{T}_0$ is emptied, the next non-empty list is used (list at position $\hat{T}_1$ in this example). The entry $\hat{T}_0$ is now used to store $T$ values quantized to $\hat{T}_0 + 6\Delta$ (circular queue). For consistency, all labels are shifted forward one position so the new $P_{min}$ is now at position $T_0$ as shown in Figure 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 (misorder), see below for more details.

The average complexity of the remove operation is $O(1)$ as long as $O(d) \leq O(n)$ (since the operation may involve searching for a non-empty queue). 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 it's run-time complexity is $O(1)$.

An extreme case occurs when $I_{max} \gg 0$ and increments with same magnitude 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 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 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" [16]. His bucket data structure is a linear array of length $O(\max T/\Delta T)$. In contrast, our approach does not require a new discretization (which in the case of [16] includes an optimization step which is more expensive to solve), uses a cyclic array thereby no needing to estimate the maximal distance and being more memory efficient, has very small constant in the $O(N)$ complexity, and quantizes the values only for queue placing and not for actual computation. Thereby, the approach here proposed is simpler, faster, valid for any dimension (as well as for the extensions reported in the literature for computing geodesics on manifolds [7,10]), and memory efficient. The authors of [8] 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*.

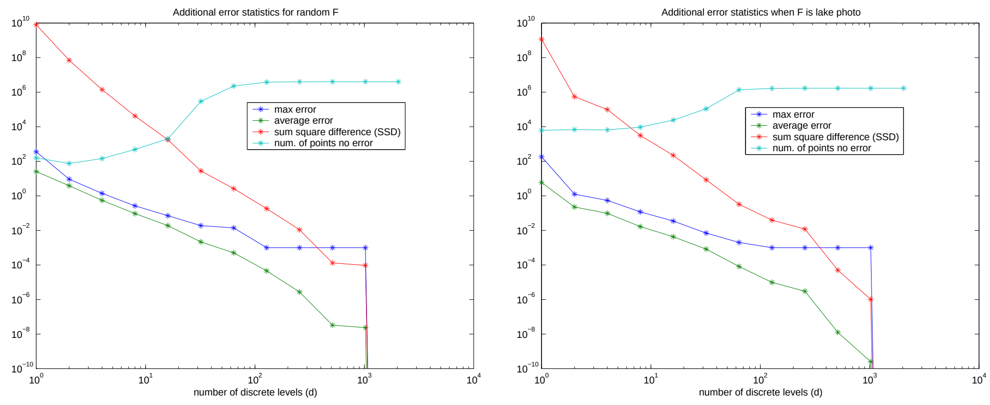
Let $T_{ro}$ be the minimal possible time value the point $P_r$ would get if $P_{min}$ was selected (the selection an accurate queue/tree would give). The additional error due to a single misordering is $\varepsilon = T_r - T_{ro}$. Since the point $P_r$ was selected, the *untidy priority queue* restricts $T_r$ and $T_{min}$ to share the same discrete level. Hence, $T_r - T_{min} \leq \Delta$. Since $F > 0$, it is possible to show from the properties of the algorithm that $T_{min} \leq T_{ro}$. When combining this, we obtain that the possible error due to a single misordering is $\varepsilon = T_r - T_{ro} \leq T_r - T_{min} \leq \Delta$. Note that $\varepsilon > 0$ since the algorithm may only decrease the value of points in $NB$, so removing a point early may only cause $T_r$ not to reach $T_{ro}$. Therefore the introduced error can not be negative.

Since $\varepsilon \leq \Delta = I_{max}/d$, by selecting $d = O(1/h)$ we can achieve $\varepsilon = O(h)$. This is the same order of magnitude of error as in the original fast marching algorithm, only that computational complexity is reduced to $O(N)$.

## 3 Numerical Experiments

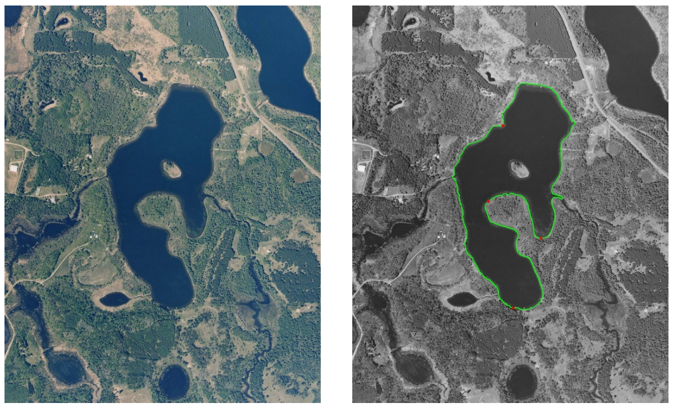
Numerical experiments show that errors caused due to the misordering are very rare, small, and in practice far from the upper bounds computed above. Figure 2 presents error statistics versus the number of discrete levels. Numerical experiments show that few discrete levels are enough to achieve a negligible error.

Figure 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



(a) (b)

Fig. 2. *Error statistics compared to an accurate queue. (a) Statistics when creating a time/distance function on a 2000x2000 grid with random* $0 < F \leq 1$. *(b) Same statistics when* $F$ *is a decreasing function of the image gradient, Figure 3.*



(a) (b)

Fig. 3. *The segmentation example shows that the quantization error does not interfere with fundamental fast marching applications. (Image size 1163* x *1463.)*

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/.

Figure 4 shows the $O(N)$ run-time complexity. The average number of queue operations done per point in the gird 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 2000x2000 grid with random $0 < F \leq 1$ using 1000 discrete bins is 1.251 seconds. When $F$ is an edge map obtained from the 1163x1463 image displayed in Figure 3(a), the time required is 0.591 seconds. The computer used for the measurements is a Pentium 4 laptop with 2Ghz processor speed.

**Acknowledgments-** We thank Gustavo Brown for ideas on queue based im-

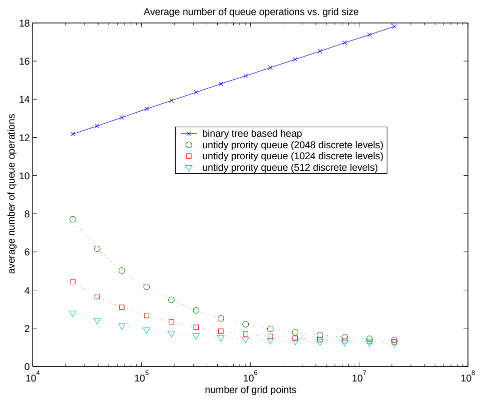


Fig. 4. *Run-time statistics. Only queue search operations are considered. For the heap operations, these include both the heap access and the priority comparisons. For the* untidy priority queue*, operations are array entry access. Statistics done on a square grid with a random $F$ function.*

plementations and Ron Kimmel for reminding us of the $O(N)$ algorithm in [16]. This work was partially supported by the Office of Naval Research, the National Science Foundation, and the National Geospatial-Intelligence Agency.

## References

[1] J. Ahn and S. Oh, "Dynamic calendar queue," *Proceedings of the Thirty-Second Annual Simulation Symposium*, pp. 20, 1999.

[2] M. Boue and P. Dupuis, "Markov chain approximation for deterministic control problems with affine dynamics and quadratic cost in the control," *SIAM J. Numer. Anal.* **36**, pp. 667-695, 1999.

[3] R. Brown,"Calandar queues: A fast O(1) priority queue implementation for the simulation event set problem," *Comm. ACM* **31**, pp. 1220-1227, 1988.

[4] L. D. Cohen, and R. Kimmel, "Global minimum for active contours models: A minimal path approach," *International Journal of Computer Vision* **24**, pp. 57-78, 1997.

[5] J. Helmsen, E. G. Puckett, P. Collela, and M. Dorr, "Two new methods for simulating photolithography development in 3D," *Proc. SPIE Microlithography* **IX**, pp. 253, 1996.

[6] R. Kimmel, *Numerical Geometry of Images: Theory, Algorithms, and Applications*, Springer-Verlag, New York, 2004.

[7] R. Kimmel and J. A. Sethian, "Computing geodesic paths on manifolds," *Proceedings of National Academy of Sciences* **95:15**, pp. 8431-8435, July, 1998.

[8] R. Kimmel and J. Sethian, "Optimal algorithm for shape from shading and path planning," *Journal of Mathematical Imaging and Vision* **14:3**, pp. 237-244, 2001.

[9] K. Leong Tan and L. Thng, "Snoopy calendar queue," *Proceedings of the 32nd Conference on Winter Simulation*, pp. 487 - 495, 2000.

[10] F. Memoli and G. Sapiro, "Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces," *Journal of Computational Physics,* **173:2**, pp. 730-764, November 2001.

[11] R. Rönngren, J. Riboe, and R. Ayani, "Lazy queue: A new approach to implementing the pending-event set," *Proceedings of the 24th Annual Symposium on Simulation*, pp. 194-204, 1991.

[12] J. A. Sethian, "A fast marching level-set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.* **93:4**, pp. 1591-1595, 1996.

[13] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision and Materials Sciences.* Cambridge Univ. Press, 1996.

[14] M. Thorup, "On RAM priority queues," *Proceedings 7th ACM-SIAM Symposium on Discrete Algorithms*, pp. 59-67, 1996.

[15] Y.R. Tsai, L.T. Cheng, S. Osher, and H.K. Zhao, "Fast sweeping algorithms for a class of Hamilton-Jacobi equations," *SIAM Journal on Numerical Analysis* **41:2**, pp. 673-694, 2002.

[16] J. N. Tsitsiklis, "Efficient algorithms for globally optimal trajectories," *IEEE Transactions on Automatic Control* **40** pp. 1528-1538, 1995.

[17] H. K. Zhao, "Fast sweeping method for Eikonal equations," *Mathematics of Computaion* **74**, pp. 603-627, 2004.