Fast and accurate geodesic distance transform by ordered propagation

Rubén Cárdenes a,*, Carlos Alberola-López b, Juan Ruiz-Alzola b, c

a Laboratory of Image Processing, University of Valladolid, Spain
b University of Las Palmas de Gran Canaria, Spain
c Canary Islands Institute of Technology, Spain

A R T I C L E   I N F O

Article history:
Received 23 January 2008
Accepted 26 May 2009

Keywords:
Distance transform
Geodesic distance transform
Geodesic metric
Hidden pixels
Ordered propagation
Visibility

ABSTRACT

In this paper, we present a new geodesic distance transform that uses a non-Euclidean metric suitable for non-convex discrete 2D domains. The geodesic metric used is defined as the shortest path length through a set of pixels called Locally Nearest Hidden Pixels, and manages visibility zones using bounding angles. The algorithm is designed using ordered propagation, which makes it extremely efficient and linear in the number of pixels in the domain. We have compared our algorithm with the four most similar geodesic distance transform techniques, and we show that our approach has higher accuracy and lower computational complexity.

© 2009 Elsevier B.V. All rights reserved.

1. Introduction

The Distance Transform (DT) of a digital image, with respect to a set of source points (or pixels) in that image, is usually called reference set, is the operation that computes for every pixel of the image the distance to the nearest pixel in the reference set. The importance of DTs is crucial in many image processing areas and image applications of the DT include sulcal segmentation [11], cortical thickness measurements [12], virtual endoscopy [13], and can include optimization problems such as route planning for robot motion [14].

One of the earliest works about GDTs was done by Lantuejoul [15] and later by Piper and Granum [9] and Verwer et al. [16]. These works have something in common: they use ordered propagation instead of raster scan propagation. The idea behind ordered propagation is that the distances from the pixels in the reference set are not propagated using masks from the corners of the image, as it is customary elsewhere [5], but rather starting from the pixels at the reference set and propagating successively their distances to their neighbors. This can be seen as wave fronts propagating at equal speed on the water surface. The simplest implementation of this metric is the geodesic version of the city block metric used by Piper and Granum [9]. Some other metrics, such as the Chamfer metric, have been used, for instance, in [9,16], but they are coarse approximations to the geodesic version of the Euclidean DT.

To the best of our knowledge the most recent and relevant works about GDTs are Cohen and Kimmel [17], Cuisenaire [18], Cárdenes et al. [19] and Coeurjolly et al. [20]. Cohen and Kimmel [17] used a fast marching approach to compute distance maps and therefore to find minimal paths in non-convex domains. Then, Cuisenaire [18] defined a new geodesic metric based on connectivity inside balls of fixed radius, in order to construct efficient implementations of the GDT using ordered propagation. Cárdenes et al. [19] proposed a GDT based on a metric through special pixels called occlusion points using them as new sources when detected. An improvement with respect to this contribution has been
proposed by Coeurjolly et al. [20]. This latter work used the same metric, but the visibility is checked in all pixels using a suitable definition of discrete visibility that allows to find the special pixels proposed by Cárdenes accurately, and the visibility zones are also managed more accurately. Although Coeurjolly et al. used ordered propagation as well, his algorithm is more cost consuming although more accurate than the others.

We propose here a new efficient method to compute the GDT using ordered propagation, with the same geodesic metric introduced in Cárdenes et al. [19], but detecting and managing more efficiently the special pixels that are used to compute the geodesic distance. Those pixels are called here *Locally Nearest Hidden Pixels* (LNHP), that will be different from the *occlusion points* obtained in [19], and that will be detected using some simple criteria (to be specified in Section 2.2). At the same time we will use the angles obtained in [19], and that will be detected using some simple criteria (to be specified in Section 2.2). At the same time we will use the angles obtained in [19], and that will be detected using some simple criteria (to be specified in Section 2.2).

We prove here that the resulting GDT is more accurate than those proposed in [17–20]. Regarding the computational complexity we demonstrate that our GDT is linear with $O(m)$, with $m$ the number of pixels in the obstacles.

This paper is organized as follows: in the next section we introduce the geodesic metric, the LNHPs and the geodesic DT. Then in Section 3 we describe the implementation details. Comparison with the techniques just mentioned is provided in terms of computational complexity (Section 4) and in terms of accuracy (Section 5). Some graphical results are shown in Section 6 and finally Section 7 presents our conclusions.

2. Geodesic metric, Locally Nearest Hidden Pixels and geodesic DT

2.1. Geodesic metric

Let $H$ denote a rectangular discrete 2D domain, or 2D grid, where its elements or pixels are 8-connected. We will consider two types of elements in $H$, namely, the domain pixels $M \subseteq H$, and the elements of the complementary set of $M$, that we will call obstacle set. We define the neighborhood of $p$, $N(p)$ of size $s$, as the pixels $q_i \in N(p)$, such as $q_i = p + n_i$, ($i = 1, \ldots, s$), and $n_i$ are the vectors that define the relative positions of the neighbors.

The general definition of the geodesic distance between two pixels can be stated as follows

**Definition 1.** The geodesic distance between two pixels $p_1$ and $p_n$ in a domain $M$ is defined as the length of the shortest path from $p_1$ to $p_n$, where the path is $P = \{p_1, \ldots, p_n\}$, and the path length $l(P)$ is defined as

$$l(P) = \sum_{i=1}^{n-1} d(p_i, p_{i+1})$$

where $p_i$ and $p_{i+1}$ are 8-connected neighbors for $i \in \{1, \ldots, n-1\}$ and $d_i$ are computed as Euclidean distances.

2.2. Visibility and Locally Nearest Hidden Pixels

In this paper, we propose to use a new definition of geodesic distance, that we will call the *geodesic distance driven by Locally Nearest Hidden (LNH) Pixels*, similarly to the idea proposed in [19]. First we will define two different zones in a non-convex domain from a given source $o$.

**Definition 2.** The visible zone of a pixel $o$, $V(o)$, is the set of pixels $p \in M$, such that for every $p$, the segment that connects $o$ with $p$, is totally included in $M$.

Therefore the hidden zone or shadow from a pixel $o$ can be defined as the complementary set of $V(o)$, that we will call $S(o)$, and consists of those pixels $p \in M$ such that the segment connecting $o$ with $p$ is not totally included in $M$. See Fig. 1.

**Definition 3.** We define a pixel $p$ as a Locally Nearest Hidden Pixel (LNHP) with respect to a source pixel $o$, if $p$ fulfills the following criteria

1. $p$ belongs to the domain, $M$.
2. $p$ has a neighbor within $N(o)$ that belongs to the obstacle set, $M$.
3. $p$ is not visible from $o$.
4. $p$ has a neighbor within $N(o)$ that belongs to $V(o)$.
5. $p$ is nearer to $V(o)$ than the pixels in its neighborhood: $N(o)$ (locally nearest).

![Fig. 1. Source, obstacles, visible, hidden and LNH pixel. Notice that although LNHPs are drawn in light blue they also belong to the hidden zone.](image-url)
We will call the above criteria as LNHP criteria. This definition follows that a LNHP is a pixel in \( S(o) \) that will have neighbors belonging to two different sets: the visible zone of \( o, V(o) \) and the obstacle set, \( M \). It is quite important to highlight that these criteria can be checked out explicitly during the propagation process, which makes the detection of LNHPs quite an efficient procedure, because it takes advantage of the information provided by the propagation process in a natural way. The fifth criterion deserves special attention: using the first four criteria it is possible to find more than one LNHP for a given source, for example \( p \) and \( r \) in Fig. 1(left) and white pixels in Fig. 1(right). The fifth criterion prevents \( r \) from being a LNHP in Fig. 1(left) because it is not nearer than \( p \) to the visible zone. Additionally the local property of this criterion allows other pixels to be LNHP if they are not in the neighborhood of other LNHP, as in Fig. 1(right).

2.3. Geodesic Distance driven by LNHPs

We now define the geodesic distance by LNHPs, which is conceptually the same as that proposed in [19] and thereafter used in [20].

**Definition 4.** The geodesic distance driven by LNHPs between \( p_1 \) and \( p_n \) in a domain \( M \), is the minimum length over all the possible polylines starting at \( p_1 \) and ending at \( p_n \), passing through the LNHPs, such that all the polylines are included in the domain \( M \).

This definition takes advantage of the fact that the geodesic path between any two pixels in the domain \( p_1 \) to \( p_n \), is a straight line if \( p_n \in V(p_1) \), or a set of segments that looks like a tight string that joins \( p_1 \) and \( p_n \), restricted to the domain.

**Definition 5.** We define the geodesic distance transform driven by LNHPs (GDT-LNHP) in a 2D rectangular grid \( H \), constrained to the domain \( M \subseteq H \), with respect to the reference set \( F \subseteq M \), as the operation that computes for every pixel in \( M \), the geodesic distance driven by LNHPs to its nearest pixel in \( F \).

3. Algorithm implementation

One efficient algorithm to implement GDTs is that of Verwer et al. [16], where a uniform cost algorithm stores the pixels in numbered buckets, as they are visited in order of increasing distance. This approach is known as bucket sorting, where the buckets are used to implement the propagation fronts. Our approach is quite similar to that bucket sorting scheme, but with some improvements, some of them described in [21]. In our method, the information related to the coordinates of the sources are propagated, and the global distance reached is updated at every new pixel, letting the propagation continue until the whole domain is visited. However, not only the coordinates of the sources are propagated, but also the information about LNHPs. For this purpose, we will store at each pixel \( p \) this information:

- \( r_{vis}(p) = r_{vis}(p), r_{vis}(p) \): the coordinates of the nearest visible source or LNHP of \( p \).
- \( d_{vis}(p) \): the geodesic distance from \( r_{vis}(p) \) to the initial source in \( F \).
- \( D(p) \): the current geodesic distance from \( p \) to the nearest source in \( F \), with floating point precision.

Using this information, when a new pixel \( p \) is reached by an adjacent pixel \( q \), the new distance is computed as follows:

\[
d_{new}(p) = d_{new}(p, r_{vis}(q)) = d_{vis}(q)
\]

that is, the Euclidean distance from \( p \) to the nearest visible source or LNHP pixel of \( q, r_{vis}(q) \), plus the distance from that LNHP to the starting source \( d_{vis}(q) \). This distance computation, initially proposed by Cárdenes et al. [19] stems directly from **Definition 4**; the geometrical interpretation is straightforward: when a LNHP is detected, it automatically becomes a new source and a new propagation front starts from it.

The two main contributions of this method are how to find the LNHPs, and how to propagate correctly the rest of pixels after a LNHP is detected. To detect a LNHP \( p \) from a source \( o \) we take advantage of the ordered propagation procedure. Using it the LNHPs from \( o \) will appear among the first non-visible pixels in the propagation of source \( o \) due to an specific obstacle. Using this fact and the criteria described before, we can detect the LNHPs very efficiently, and manage correctly the pixels propagated after them.

The pseudo code of the GDT algorithm proposed here is shown in Algorithms 1–4. In Algorithm 1 we show a bucket sorting scheme, as mentioned above, but after a bucket is emptied, we extract the elements of an auxiliary list, called listsaux, where we have stored the neighbors of tentative LNHPs. They are then propagated using the procedure Propagate to candidates.

In Algorithm 2 the procedure propagate is shown. This procedure takes as inputs the pixel \( p \) to be propagated, and the current integer distance \( d \), and it checks the eight connected neighbors of \( p, \ u \in N(p) \). First, the new distance at \( u, d_{new} \) is computed using Eq. (2), as well as the rounded integer distance, \( d_{aux} = \text{round}(d_{new}) \). Then, we distinguish three different cases:

- **Case 1:** The pixel \( p \) is outside the domain, \( i.e., p \notin M \). In that case, we put \( p \) in listsaux, to be propagated using the procedure Propagate to candidates. In all the other cases the pixel \( p \) belongs to \( M \).
- **Case 2:** The new pixel has not been visited \( (D(u) = -1) \) and \( d_{aux} \geq d \). In this case we update \( r_{vis}(u) \) and \( d_{vis}(u) \) with the same values as in \( p \), and \( D(u) \) with the new distance \( d_{new} \). The new pixel is put in bucket \( |d_{aux}| \).
- **Case 3:** The new pixel has been visited \( (D(u) = -1) \), its integer distance is greater than the current distance \( (d_{aux} > d) \), the sources of \( p \) and \( u \) are different \( (r_{vis}(p) \neq r_{vis}(u)) \), and \( u \) belongs to \( M \). This situation occurs at boundaries of different visibility zones, because the sources of pixels \( p \) and \( u \) are different. Then if the source of \( u \) is the same as the source of \( p \), that means that we are trying to propagate a pixel \( p \) that belongs to a visibility zone to a pixel \( u \) that was previously marked in a contiguous visibility zone. Therefore we need to check if the new pixel \( u \) is visible from its initial source using the angle \( \alpha \) of the segment joining \( u \) and its source, and checking if \( \alpha \) lies between the minimum and maximum angle allowed for the obstacle that generated the LNHP pixel located at \( r_{vis}(p) \). If \( \alpha \) is between those angles \( u \) is recomputed as it comes from \( r_{vis}(p) \). This case corrects pixels marked as visible from a source pixel but that are not actually visible from it, but they lie in the shadow of an obstacle. This situation is illustrated in Fig. 2, where we show a GDT with the propagation stopped at distance \( d = 8 \). At his time, if pixel \( u \) (that should belong to the shadow), is accessed first by pixel \( s \), \( u \) will be erroneously marked as visible. Then pixel \( p \) should correct this situation when it visits \( u \), recomputing its value and marking it as coming from the LNHP.

Notice that the procedure propagate allows pixels to be partially propagated outside the domain. When any of those pixels in \( M \) are propagated to any pixel \( p \) in \( M \), we use a different propagation procedure called Propagate to candidates, shown in Algorithm 3. In this procedure we check if any pixel neighbor to \( p \) inside the domain can be a LNHP. We show in Fig. 3 an example to explain this procedure. Consider \( p \), the leftmost obstacle pixel in Fig. 3. When procedure 3 is executed at this pixel, four candi-
dates for LNHP are found in our example (see Fig. 3(a)) using the
condition of line 3, of Algorithm 3. It is obvious that those pixels
fulfill the first two LNHP criteria. It is also easy to see that pixels
already visited and marked with a distance equal to the distance
of the obstacle are visible by the source, so this condition is used
implicitly for efficiency to discard visible pixels that cannot be
candidates to LNHP due to the third LNHP criteria. The condition
at line 6 discards the rest of the pixels that are visible from the
source, obtaining candidates in Fig. 3(b). Then, condition at line
10 discards pixels that are not neighbors to the visible zone, by
the fourth LNHP criteria, obtaining Fig. 3(c) and condition at line
13 chooses the nearest candidate to the visible zone by the fifth
LNHP criteria, obtaining the final candidate in Fig. 3(d).

Once a LNHP is detected, we store it in bucket[d] to be prop-
gagated later, and we compute the maximum and minimum an-
gles allowed for the pixels that will be propagated from it.
This angle will be computed using the most limiting corner
angles allowed for the pixels that will be propagated from it. In practice, we will only use one of those
two angles, because shadows are bounded by different LNHPs, or
by the domain bounds. In order to compute accurately the dis-
tances, we compute the maximum and minimum angles
by the domain bounds.

\[ d_{\text{new}} = D(r_{\text{src}}(p)) + d_{\text{int}}(r_{\text{src}}(p), cp) + d_i(cp, s) \] (3)

Algorithm 1. Geodesic distance transform driven by LNH pixels
(GDT-LNHP)

**Input:** Reference set \( F \), from where we want to compute the
geodesic DT, and the 2D grid \( H \), where foreground pixels belong
to the domain \( M \), and background pixels belong to the obstacles \( \overline{M} \).

**Output:** Geodesic DT: \( D \) from the reference set \( F \), restricted to \( M \).
1: Initialize: \( D(p) = - \infty \) \( \forall p \in M \) and \( d_{\text{int}}(p) = 0 \) \( \forall p \in F \)
2: Put the elements of \( F \) in bucket[0]
3: \( d = 0 \)
4: while there are no more pixels in the buckets do
5: \quad while(bucket[d] \( \neq \phi \)) do
6: \quad Get \( p \) from bucket[d]
7: \quad propagate(\( p, d \))
8: \quad end while
9: \quad while(listaux \( \neq \phi \)) do
10: \quad Get \( p \) from listaux
11: \quad Propagate to candidates(\( p \))
12: \quad end while
13: \( d = d + 1 \)
14: end while

Algorithm 4 shows the procedure to check the visibility be-
tween two pixels given another possible obstacle pixel. To this
end we compute the maximum and minimum angles \( \beta_{\text{max}} \) and \( \beta_{\text{min}} \)
that limit the visibility of one pixel due to the obstacle, and then if
the angle \( \alpha \) of the line between the two pixels is between \( \beta_{\text{max}} \) and
\( \beta_{\text{min}} \) it will be non-visible, see Fig. 4.

It is important to highlight some issues: first, integer distance
values are used to order the propagation fronts, but the real dis-
tance values are obtained and stored for every pixel. It is also
important to notice that, if the domain is convex LNHPs do not ex-
ist, therefore Eq. (2) reduces to the Euclidean distance. Notice fin-
ally that some minor details of the algorithm have been omitted
in order to simplify the description.

**Algorithm 2. Propagate**

1: \quad procedure propagate(\( p, d \))
2: \quad for all \( u \in N_G(p) \) do
3: \quad \( d_{\text{new}} = \text{dist}(u, r_{\text{src}}(p)) + d_{\text{int}}(p) \)
4: \quad \( d_{\text{int}} = \text{round}(d_{\text{new}}) \)
5: \quad if \( (p \in \overline{M}) \) then \( \text{case 1} \)
6: \quad put(\( p \) in listaux)
7: \quad else
8: \quad \quad if(\( d_{\text{int}} > d \) and \( D(u) = \infty \)) then \( \text{case 2} \)
9: \quad \quad \quad \quad \( d_{\text{int}}(u) = d_{\text{int}}(p) \)
10: \quad \quad \quad \quad \( r_{\text{src}}(u) = r_{\text{src}}(p) \)
11: \quad \quad \quad \quad \text{put(\( u \) in bucket } [d_{\text{int}}] \)
12: \quad \quad \quad \quad \text{D(}u\text{) = }d_{\text{int}}\)
13: \quad \quad \quad \text{end if}
14: \quad \quad \quad \text{if(}D(u) = \infty \) and \( d_{\text{int}} > d \) and \( r_{\text{src}}(u) = r_{\text{src}}(p) \) and \( u \in M \)
15: \quad \quad \quad \text{then } \text{case 3}
16: \quad \quad \quad \text{\( \alpha = \arctan(\frac{y-y_{r_{\text{src}}(u)}}{x-x_{r_{\text{src}}(u)}}) \)
17: \quad \quad \quad \text{if(} \alpha < \beta_{\text{min}}(r_{\text{src}}(p)) \) and \( \alpha > \beta_{\text{max}}(r_{\text{src}}(p)) \) then
18: \quad \quad \quad \quad \text{\( d_{\text{int}}(u) = d_{\text{int}}(p) \)
19: \quad \quad \quad \quad \text{\( r_{\text{src}}(u) = r_{\text{src}}(p) \)
20: \quad \quad \quad \quad \text{put(\( u \) in bucket } [d+1] \)
21: \quad \quad \quad \text{end if}
22: \quad \quad \quad \text{end if}
23: \quad \quad \text{end if}
24: \quad \quad end if
25: \quad end for
26: end procedure

4. Computational complexity

In Fig. 5 we show the computational times in logarithmic scale
obtained from five different GDT algorithms, versus the number
of pixels in the domain. The algorithms are our GDT-LNHP, the Coeur-
jolly algorithm [20], the fast marching approach given in [17] the B3-
GDT by circular propagation, (using ball sizes of radius \( d = 2 \) and
\( d = 6 \)) [18], and the OPPGDT algorithm [19]. The execution times
shown correspond to domain sizes from \( 0.5 \times 10^5 \) to \( 4.5 \times 10^5 \) pix-
els, and have been measured in a Linux server with an 2.6 GHz Dual
Core AMD Opteron processor.

The computational complexity of our GDT derives from the
number of operations made at each pixel (distance computa-
tions, assignments, high comparisons) as well as the number of
times each pixel is visited during the propagations. Procedure
propagate is called a number of times equal to the number of
pixels \( n \) in the domain \( M \), corresponding to pixels accessed
for the first time, plus a number of times \( z \), that stems from
Author's personal copy

Let $T_1$ be the time taken in the worst case for this procedure (case 3), which is limited by a fixed number of operations.

Algorithm 3. Propagate to candidates

1: procedure Propagate to candidates ($p$)
2: Find all $s_i \in M$ neighbors to $p$, such as:
3: $D(s_i) = -1$ OR $round(D(s_i)) = round(D(p)) + 1$
4: if ($s_i = \emptyset$) then
5: Find all $s_j \in s_i$ such as:
6: check-visibility ($s_j, p, rsrc(p)$) = 0
7: end if
8: if ($s_j = \emptyset$) then
9: Find all $s_k \in s_j$ such as:
10: $s_k$ has a visible neighbor ($q$)
11: end if
12: if ($k > 1$) then
13: Find the $s \in s_k$ closest to the visible zone
14: end if
15: Find most limiting corner of $p$: ($c_{pr}, c_{py}$)
16: $\alpha_1 = arctg \frac{c_{py} - rsrcy(p)}{c_{pr} - rsrcx(p)}$
17: $\alpha_2 = arctg \frac{c_{py} - rsrcy(s)}{c_{pr} - rsrcx(s)}$
18: if ($\alpha_1 < \alpha_2$) then
19: $\alpha_{min}(s) = \alpha_1$ and $\alpha_{max} = +\infty$
20: else
21: $\alpha_{max}(s) = \alpha_1$ and $\alpha_{min} = -\infty$
22: end if
23: $d_{new} = D(rsrc(p)) + d_k(rsrc(p), cp) + d_k(cp, s)$
24: $d_{src}(s) = d_{new}$
25: $rsrc(s) = s$
26: Put $s$ in bucket[$round(d_{new})$]
27: $D(s) = d_{new}$
28: $D(p) = -2$
29: end procedure

On the other hand, the procedure Propagate to candidates is called a number of times equal to $s$, the number of pixels in $M$ adjacent to $M$. Let $T_2$ be the time taken in the worst case in this function, which is also bounded by a fixed amount of operations. The overall execution time $T$ will be bounded by

$$T \leq T_1(n+z) + T_2s < \max(T_1, T_2)(n+z+s) < \max(T_1, T_2)2n$$ (4)
Algorithm 4. Check-visibility

1: procedure check-visibility (s, obst, src)
2: \( \alpha = \arctan \frac{sy}{sx} \)
3: \( \beta[1] = \arctan \frac{obsty}{obstx} \)
4: \( \beta[2] = \arctan \frac{obsty}{obstx} \)
5: \( \beta[3] = \arctan \frac{obsty}{obstx} \)
6: \( \beta[4] = \arctan \frac{obsty}{obstx} \)
7: \( \beta_{\text{max}} = \max(\beta[i]) \quad i = \{1, 2, 3, 4\} \)
8: \( \beta_{\text{min}} = \min(\beta[i]) \quad i = \{1, 2, 3, 4\} \)
9: if \( \alpha > \beta_{\text{max}} \) OR \( \alpha < \beta_{\text{min}} \)
10: return 1 (obst is not an obstacle between s and src)
11: else
12: return 0 (obst is an obstacle between s and src)
13: end if
14: end procedure

Notice that \( z + s < n \) because \( Z \subseteq M, S \subseteq M \) and \( Z \cap S = \emptyset \). Therefore, our algorithm is linear with the number of pixels in the domain \( O(n) \), as it is shown in Fig. 5, but it is slightly slower than other \( O(n) \) methods such as the OPPGDT and the Bd-GDT because it needs more number of operations per pixel.

Regarding the difference between our algorithm and the one proposed in [20], it is clear that our method is computationally more efficient as shown in Fig. 5. The main reason of this behavior is because in [20] it is necessary to check the visibility for every pixel, as well as a structure to store the obstacles ordered by polar angles, whereas in our case only special pixels are checked (those suspected to be LNHPs), no ordered structures are needed, and only special care has to be taken in the boundaries between visibility and hidden zones.

5. Accuracy study

We have carried out an experiment in a synthetic \( 256 \times 256 \) domain with several obstacles. In such a domain we have computed analytically the distance values for all the pixels of the image except those in the obstacle, to the upper left corner pixel. With that reference image we have computed the average and maximum error for the same GDT algorithms compared before. We illustrate in Fig. 6 the absolute difference images between every GDT and the reference image using warm colors for higher error values. Notice that the color-map represent values between 0 and 2 in all figures except in (e) and (f) where the color-map show values between 0 and 10. In Table 1 we also summarize the numeric error values.

The algorithm here proposed shows the best error figures; regarding to the second algorithm in the table, it shows higher error rates and it worsens in areas where distances are larger (see Fig. 6(b)); this is due to the fact that small errors in the computation of the distance at the beginning of occlusion zones increase with the distance and with the number of obstacles. The GDT-LNHP performs better in terms of accuracy because...
those distances are more accurately computed using the corner of the obstacles, (see Fig. 6(a)). The mean error increases for the \( d = 6 \) Bd-GDT, although the maximum error value is lower in this case. However, the Bd-GDT produces sharper results, as it can be seen in Fig. 6(c). The following algorithm in increasing order is the OPPGDT, shown in Fig. 6(d). Notice that although the Bd-GDT with \( d = 6 \) and the OPPGDT have very similar mean error values, the Bd-GDT with \( d = 6 \) is much smoother and its maximum error value is significantly lower, making the Bd-GDT with \( d = 6 \) preferable in terms of accuracy. Finally, the fast marching approach and the Bd-GDT (\( d = 2 \)) have very high error values, which make them poor solutions for the GDT.

### 6. Results

In order to visually compare the results of different GDT algorithms, we show in Fig. 7 the results of the methods compared before. The color-map used is a cyclic grayscale color-map, where a set of increasing gray values are repeated every 10 units. This representation is used to see the local details of the DT. Notice that the best algorithms in terms of accuracy (our GDT-LNHP, Coeurjolly and Bd-GDT with \( d = 6 \)), are shown in the first row, and are difficult to compare visually, whereas the rest of the methods have clear differences with respect to the others. The OPPGDT has undesired corner effects after the obstacles, the iso-contours of the fast marching are not completely circular, and the iso-contours of the Bd-GDT with \( d = 6 \) and \( d = 2 \), are approximations of Euclidean distances, and their iso-contours near the object have also octagonal shape.

We show in Fig. 8 our GDT-LNHP computed at a simple domain from a single source pixel. We show in the middle of Fig. 8 the shadows projected from each LNHP and in the right, the shadows produced by each obstacle. Notice that our method distinguishes between different hidden zones, and therefore visibility zones are correctly computed in order to obtain the shortest distance at every pixel.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Max error</th>
<th>Average error</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDT-LNHP</td>
<td>1.084</td>
<td>0.445</td>
</tr>
<tr>
<td>GDT Coeurjolly</td>
<td>2.766</td>
<td>0.552</td>
</tr>
<tr>
<td>GDT-Bd ( d = 6 )</td>
<td>2.573</td>
<td>0.721</td>
</tr>
<tr>
<td>OPPGDT</td>
<td>5.498</td>
<td>0.789</td>
</tr>
<tr>
<td>Fast marching</td>
<td>13.869</td>
<td>7.255</td>
</tr>
<tr>
<td>GDT-Bd ( d = 2 )</td>
<td>19.441</td>
<td>7.466</td>
</tr>
</tbody>
</table>

Fig. 7. GDT using several algorithms, (a) GDT-LNHP, (b) Coeurjolly, (c) Bd-GDT \( (d = 6) \), (d) OPPGDT, (e) fast marching and (f) Bd-GDT \( (d = 2) \).

Fig. 8. GDT-LNHP in a domain with obstacles from a single source, coded using a cyclic grayscale color-map (left), hidden zones associated to each LNHP pixel (middle) and hidden zones associated to each obstacle (right).
In Fig. 9 we show the GDT from different non convex domains. The first three rows are synthetic non convex domains and the last row shows a domain from an axial view of the gray matter of a human brain. The second and the third rows show cases where the reference set consists of several source pixels (three and six, respectively). This figure is represented in two color-maps. The color-map in the left column represents low distances with cold colors, and high distances with warm colors. This representation is specially useful to see the global distances from the sources. The color-map in the right column is a cyclic grayscale color-map.

In Fig. 10(a) we show our geodesic DT in another domain, where the distances are computed starting from a single source located near the left upper corner of the image. We also show in Fig. 10(b) the geodesic path computed from a pixel located near the bottom of the image, where the geodesic path goes through the LNHPs detected. The LNHPs are also shown in Fig. 10(c) (white pixels), and Fig. 10(d) shows the hidden zones associated to each LNHP.

7. Conclusions

In this work, we have presented a new algorithm to compute GDTs in 2D discrete domains, showing that it has a computational complexity linear in the number of elements in the domain $O(n)$. In order to validate our results we have compared in terms of accuracy and computational load, with the four closest algorithms existing in the literature, showing that our technique has the highest accuracy, and its computational complexity is close to other simpler GDT algorithms as shown in Fig. 5, and it outperforms the second more accurate GDT algorithm, proposed in [20].
We have also shown that ordered propagation is crucial in our algorithm due to several issues. First, the LNHPs detection can be carried out very efficiently using the propagation process. Second, it is possible to distinguish visibility zones coming from different sources using the maximum and minimum angles at every LNHP, and it is done very fast. Third, the number of operations is linear with $n$, and finally, it is possible to account for several source pixels in the reference set at the same time.

Acknowledgments

The authors acknowledge the Comisión Interministerial de Ciencia y Tecnología for research grants TEC 2004-06647-C03-01 and TEC 2007-67073/TCM, the Fondo de Investigaciones Sanitarias for grant PI-041483, the Junta de Castilla y León for grant VA075A05 and the European Commission for the funds associated to the Network of Excellence SIMILAR (FP6-507609). The first author also acknowledges Emma Muñoz for her helpful comments about this work.

References


