TOLERANCE-BASED FEATURE TRANSFORMS

Dennie Reniers, Alexandru Telea
Department of Mathematics and Computer Science, Eindhoven University of Technology
Den Dolech 2, 5600 MB, Eindhoven, The Netherlands
D.Reniers@tue.nl, alext@win.tue.nl

Keywords: feature transform, tolerance-based, distance transform, algorithms, image processing.

Abstract: Tolerance-based feature transforms (TFTs) assign to each pixel in an image not only the nearest feature pixels on the boundary (origins), but all origins from the minimum distance up to a user-defined tolerance. In this paper, we compare four simple-to-implement methods for computing TFTs on binary images. Of these methods, the Fast Marching TFT and Euclidean TFT are new. The other two extend existing distance transform algorithms. We quantitatively and qualitatively compare all algorithms on speed and accuracy of both distance and origin results. Our analysis is aimed at helping practitioners in the field to choose the right method for given accuracy and performance constraints.

1 INTRODUCTION

A distance transform (DT) computes, for pixel p of an image, the distance $D(p) = \min_{q \in \delta\Omega} \|q - p\|$ to the nearest feature pixel, or *origin*, q on the boundary $\delta\Omega$ of some object Ω located in the image (Fig. 1). Non-object pixels will be denoted by $\bar{\Omega}$. There can be several equidistant origins q for a pixel p, i.e., the origin set $S(p) = \arg\min_{q \in \delta\Omega} \|q - p\|$ of p can have more than one element. The feature transform (FT) assigns to each pixel p the origin set $S(p)$. A *simple* FT computes only one origin per pixel, which is sufficient for some applications. We define a *tolerance-based* FT (TFT) as a map that assigns to each pixel the origin set $S_\epsilon(p) = \{q \in \delta\Omega \mid \|q-p\| \leq D(p)+\epsilon\}$, where ϵ is a user-defined tolerance. One use of the TFT is to compute exact Euclidean DTs, as first observed in (Mullikin, 1992) (see also Sec. 5). A second use of the TFT is to compute robust, connected skeletons or medial axes, as follows. Medial axis (MA) points can be detected as the FT points having at least two origins, one on each side of the axis (Foskey et al., 2003). However, this definition may yield disconnected skeletons in a discrete space. For example, for a rectangle of even height, no pixels lie exactly in the middle. Using a TFT with tolerance 1, origins from both axis sides are found, yielding a connected skeleton. Figure 2 shows the TFT for an object using four different tolerances ϵ. The pixel intensity denotes the origin set size $|S_\epsilon|$. The origin sets of four selected pixels are shown using white line segments. For $\epsilon = 0$, it can be seen that pixel p has only one origin because the horizontal rectangle is of even height. Using a tolerance $\epsilon \geq 1$, the origin set $S_\epsilon(p)$ contains origins from both sides of the rectangle.

Figure 1: The distance transform of an object. The pixel intensity denotes distance to the object boundary.

Overall, both DTs and FTs have numerous applications in many domains (Cuisenaire, 1999; Ye, 1988) ranging from image processing, pattern recognition, and shape representation and modeling, to path planning, computer animation, skeletonization, and optimization algorithms.

DT and FT algorithms can be classified by the order in which they process the image pixels. Raster scanning algorithms (Danielsson, 1980) sequentially process pixels in scan-line order, needing multiple passes

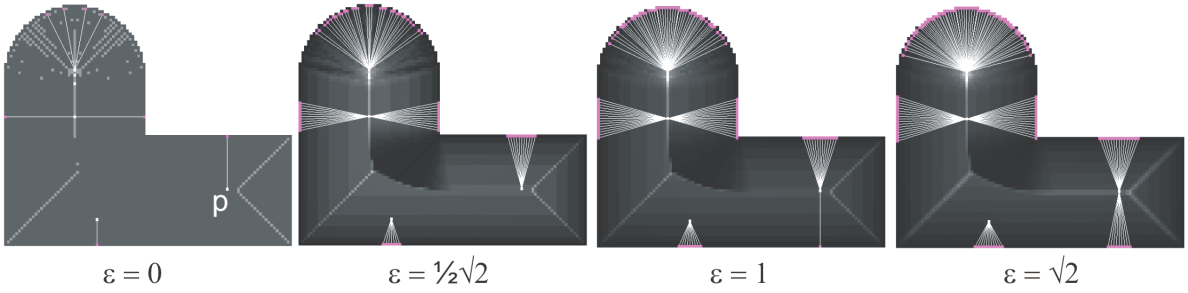

Figure 2: The TFT of an object using four different tolerances $\epsilon = 0, \frac{1}{2}\sqrt{2}, 1, \sqrt{2}$.

in which pixels are assigned new minimum distances. Ordered propagation methods (Ragnemalm, 1992) reduce the number of distance computations needed by updating only pixels in a contour set, which propagates from the object boundary $\delta\Omega$ inwards. Ordered propagation methods accommodate (distance-based) stopping criteria easier than raster scanning ones, thus being more efficient for some applications. A well-known class of ordered propagation methods are level-set and fast marching methods (FMM) (Sethian, 1999), which evolve the contour $\delta\Omega$ under normal speed (see Section 2). Although the FMM does not compute an exact Euclidean DT, the speed function it uses can be locally varied to compute more complex DTs, e.g., anisotropic, weighted, Manhattan, or position-dependent ones (Sethian, 1999; Strzodka and Telea, 2004). Recent FMM extensions compute an FT (Telea and van Wijk, 2002; Telea and Vilanova, 2003). However, this is only a simple FT, and can be quite inaccurate in many cases. Applications using this origin set, such as skeletonization, can deliver wrong results, as pointed out in (Strzodka and Telea, 2004).

As the above outlines, DT and FT methods have many, often subtle, trade-offs, which are not obvious to many practitioners in the field. In this paper, we discuss several competitive DT, FT, and TFT methods. Some of these methods extend existing ones, while others are new. We quantitatively and qualitatively compare the results of all methods with the exact TFT computed by brute force, and discuss the computational advantages and limitations of every method. The goal of our analysis is to provide a quantitative, practical guideline for choosing the "right" DT or (T)FT method to best match real-world application requirements, such as precision, performance, completeness, and implementation complexity.

This paper is structured as follows. In Section 2 we discuss the FMM and we detail on its inaccuracies. In Section 3, we modify the existing Augmented Fast Marching Method (AFMM) to yield exact simple FTs, and illustrate its use by skeletonization applications. In Section 4, we extend this idea to compute TFTs by adding a distance-to-origin tolerance. In Section 5, we analyze Mullikin's raster scanning DT, and get insight into how to set our TFT tolerance to compute exact DTs. In Section 6, we present a novel method, called ETFT, based on a different propagation order than the FMM. In Section 7, we compare our new ETFT with the related graph-search method of (Lotufo et al., 2000), and also extend the latter to compute TFTs. Finally, we quantitatively compare all of the above methods (Sec. 8) and come to a conclusion (Sec. 9).

2 FAST MARCHING METHOD (FMM)

Level set methods are an Eulerian approach for tracking contours evolving in time. The fast marching method (FMM) (Sethian, 1999) treats the special case of contours with constant sign speed functions F. The contour position p, given by its arrival time $T(p)$, is the solution of $\|\nabla T\|F = 1$ with $T = 0$ for the initial contour $\delta\Omega$. If $F = 1$, we obtain the Eikonal equation $\|\nabla T\| = 1$ whose solution is the Euclidean DT of $\delta\Omega$. The FMM efficiently computes T using the fact that $T(p)$ depends only on the T values of p's neighbors $N(p)$ for which $T(N(p)) < T(p)$. The FMM builds T from the smallest computed T values by maintaining the pixels in the evolving contour, or narrow band, sorted on T. Pixels are split into three types: *known* pixels p^{K} have an already computed T; *temporary* pixels (p^{T}) have a T subject to update; and *unknown* pixels (p^{U}) have not yet been assigned a T value. Invariant is $T(p^{\mathrm{K}}) \leq T(p^{\mathrm{T}}) \leq T(p^{\mathrm{U}})$. Initially, all pixels on $\delta\Omega$ are known to have $T = 0$ and their neighbors become temporary. Next, the temporary pixel p with smallest T becomes known (as its T cannot be influenced by other pixels), its unknown neighbors $N^{\mathrm{U}}(p)$ become temporary, and their T values are updated based on their own known neighbors, until all pixels become known. For a contour of length $B = |\delta\Omega|$ and area $N = |\Omega|$ pixels, the FMM needs $O(N \log B)$ steps, because it visits each object pixel once, and keeping the narrow band sorted on T in

Table 1: Differences between (approximate) FMM and exact Euclidean distances D ($\%e$: ratio of erroneous pixels to object pixels; $\max e$: maximum error; $\bar{e}$: average error).

image	img.size	$\max D$	$\%e$	$\max e$	$\bar{e}$
bird	238×370	49.82	89%	0.679	0.142
leaf	410×444	70.63	84%	1.013	0.290
dent	464×397	134.06	15%	1.210	0.082

each iteration needs $O(\log B)$ steps.

The DT computed by the FMM is not exact. Errors occur due to the approximation of the gradient ∇T, usually of first or second order, the former being the most common. The errors are accumulated during the propagation. In (Sethian, 1999, Sec. 12.3), the FMM accuracy is briefly treated, but no comments are made on the implications for real-world applications. Figure 3 and Table 1 show the difference between the FMM and the exact DT for some typical shapes. High errors (bright areas in Fig. 3) seem to "diffuse" away from boundary concavities. Indeed, a temporary point at a narrow band concavity has just one known neighbor N^{K}, so its distance is updated from a single known $T(N^{\mathrm{K}})$ value. A point at a narrow band convexity has several known neighbors, so its distance T is updated using more information. The maximal DT error can easily exceed 1 pixel (cf. Table 1), and can grow arbitrarily with the image size.

3 AFMM STAR

In (Telea and van Wijk, 2002), the Augmented FMM (AFMM) is presented, which computes one origin per pixel by propagating an arc-length parameterization U of the initial boundary $\delta\Omega$ together with FMM's T value. $U(p)$ basically identifies an origin $S(p)$. When a narrow band pixel p is made known and its unknown direct 4-neighbors $a \in N_4^{\mathrm{U}}(p)$ are added to the narrow band, $U(a)$ is set to $U(p)$. After propagation has completed, for all points q where U varies with at least τ over $N_4(q)$, a segment of at least length τ from the original boundary collapses. Hence, the above point set $\{q\}$ represents a (pruned) skeleton, or medial axis, of $\delta\Omega$, with τ as the pruning parameter. AFMM's complexity remains the same as for the FMM, namely $O(N \log B)$, where N is the number of object pixels and B the boundary size, because it adds just the propagation of one extra value U. Similar methods are (Costa and Cesar, 2001) for digital images and (Ogniewicz and Kübler, 1995) for polygonal contours, respectively.

However efficient and effective for computing simple FTs and skeletons, the AFMM has several accuracy problems when computing U, as can be easily seen from the resulting skeletons. Errors show up as skeleton branches having the wrong angle, are too thick, or are disconnected (e.g., Fig. 4 left). The reason is that the value $U(a)$ is determined by only *one* pixel $p \in N_4(a)$, namely the p that is first made known. We propose to solve this problem as follows. For a point a that is made temporary, we set $S(a)$ (or equivalently $U(a)$) to the closest origin among the neighbor's origin sets, i.e.:

$$S(a) = \underset{q \in S(N_8^{\mathrm{K},\mathrm{T}}(a))}{\arg\min} \|q - a\|. \qquad (1)$$

This method, which we call AFMM Star, solves AFMM's inaccuracy problems, i.e., yields a reliable simple FT method. AFMM Star robustly computes pixel-exact, pixel-thin, connected skeletons for arbitrarily complex noisy 2D boundaries (e.g., Fig. 4 right). One remaining problem is that we use the numerically inexact FMM DT (cf. Sec. 2). For practical applications, e.g. skeletonization, incorrect skeleton points will occur only where the FMM DT error exceeds 1 pixel. From Table 1, we see that this happens only at a very few pixels of relatively large objects. This gives a quantitative estimate of the AFMM Star limitations.

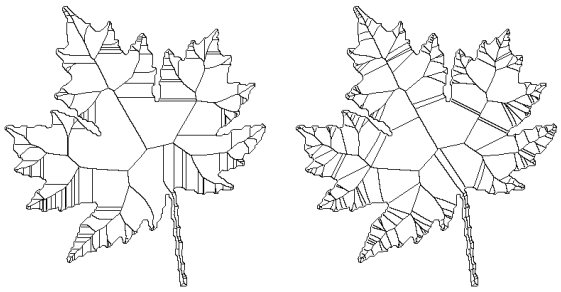

Figure 4: AFMM skeletonization errors (left). AFMM Star skeletonization (right).

4 FAST MARCHING TFT

The AFMM Star is a simple FT, i.e., it computes just one origin per point. However, some applications, such as angle-based skeletonization (Foskey et al., 2003) require all origins to be found. Moreover, multiple-origin FTs are desired for the reasons outlined in Section 1.

We now propose the novel Fast Marching TFT (FMTFT) which computes for each pixel p an origin set S_ϵ whose size depends on a user-defined distance tolerance ϵ, i.e.:

$$S_\epsilon(p) = \left\{ q \in \delta\Omega \;\middle|\; \|q - p\| \le D(p) + \epsilon \right\}. \qquad (2)$$

The pseudo code is shown in Figure 5. The distances D^{f} are computed by the FMM (line 15), see

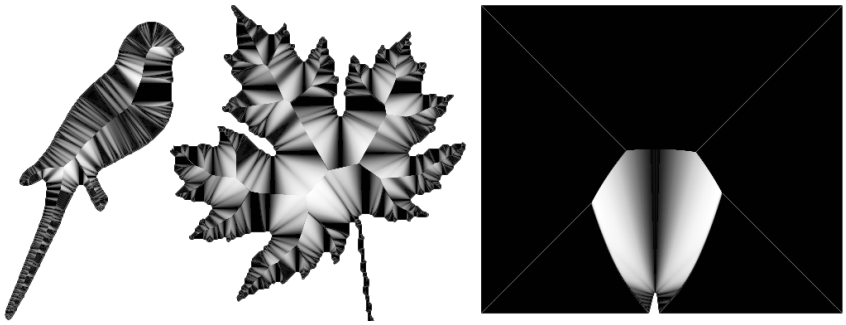

Figure 3: Differences between the (approximate) FMM distance and the exact Euclidean distance for the 'bird', 'leaf', and 'dent' images. Black indicates no error, white indicates the maximum error. See Table 1 for the exact values.

e.g. (Sethian, 1999). We initialize the origin set $S(p) = \{q \in \delta\Omega \mid \|q-p\| \leq \epsilon\}$ for $p \in \delta\Omega$. When the distance of a point a is updated during the FMM evolution, we simultaneously construct a candidate set C (line 16):

$$C(a) = \bigcup_{q \in N_s^{\mathrm{K},\mathrm{T}}(a) \cup \{a\}} S(q), \qquad (3)$$

where s is the neighborhood size. Next, let the distance $D(a) = \min_{q \in C(a)} \|q - a\|$. $D(a)$ is more accurate than the FMM distance $D^{\mathrm{f}}(a)$, because it is computed directly as the distance from a to its nearest origin, while D^{f} is computed incrementally by a first-order approximation of the gradient. D^{f} is used only to determine the propagation order (line 11), as for the AFMM Star (Sec. 3). The tolerance-based origin set $S(a)$ is constructed by *pruning* C in line 18:

$$S_\epsilon(a) \leftarrow \left\{ q \in C \;\middle|\; \|q - p\| \leq D(p) + \epsilon \right\}. \qquad (4)$$

Thus, this algorithm assumes that the origin set of a pixel a can be determined from the origin sets of a's neighbors. Statement (4) also occurs in all other to-be-discussed methods (line 30 in Fig. 7, line 17 in Fig. 8, and line 16 in Fig. 9).

The accuracy of D is influenced by the neighborhood size s and the tolerance ϵ. D can be made more accurate by increasing s. In general however, D cannot be made exact no matter the choice of s. This is because the Voronoi regions of origin pixels are not always connected sets on a discrete grid (Cuisenaire, 1999). In contrast, ϵ *can* be set so that all distance errors are eliminated. This was also observed by Mullikin, in a related context, as detailed in the next section.

```
 1: for each p ∈ Ω ∪ Ω̄ do
 2:     if p ∈ Ω̄ then
 3:         f(p) ← K, D^f(p) ← −1
 4:     else if p ∈ δΩ then
 5:         f(p) ← T, D^f(p) ← 0, S(p) ← {q ∈ δΩ | ‖q − p‖ ≤ ε}
 6:     else if p ∈ Ω ∧ p ∉ δΩ then
 7:         f(p) ← U, D^f(p) ← ∞
 8:     end if
 9: end for
10: while ∃_q f(q) = T do
11:     p ← arg min_{q:f(q)=T} D^f(q)
12:     f(p) ← K
13:     for each a ∈ N_4^{U,T}(p) do
14:         f(a) ← T
15:         D^f(a) ← min(D^f(a), compdist(N_4^K(a)))
16:         C ← ⋃_{q ∈ N_s^{K,T}(a) ∪ {a}} (S(q))
17:         D(a) ← min_{q∈C} ‖q − a‖
18:         S(a) ← {q ∈ C | ‖q − a‖ ≤ D(a) + ε}
19:     end for
20: end while
```

Figure 5: Fast Marching TFT (FMTFT).

5 ϵ-VECTOR DISTANCE TRANSFORM

Mullikin presents in (Mullikin, 1992) a scan-based algorithm for computing exact Euclidean DTs. He first identifies pixel arrangements for which Danielsson's scan-based vector distance transform (VDT) with 4-neighborhoods (Danielsson, 1980) yields inexact distances. The problem of the VDT is that it stores only one origin. In Figure 6, the VDT computes that $S(q) = \{a\}$ and $S(r) = \{b\}$. For p, one of the nearest origins from its 4-neighbors is taken. Thus $S(p) \in S(q) \cup S(r) = \{a, b\}$, while the actual nearest origin is $S(p) = \{c\}$. This situation occurs when there are three object pixels a, b, c so that $\|\vec{aq}\| < \|\vec{cq}\|$, $\|\vec{br}\| < \|\vec{cr}\|$, $\|\vec{cp}\| < \|\vec{ap}\|$, and $\|\vec{cp}\| < \|\vec{bp}\|$, i.e., when the hatched area contains a grid point. Mullikin

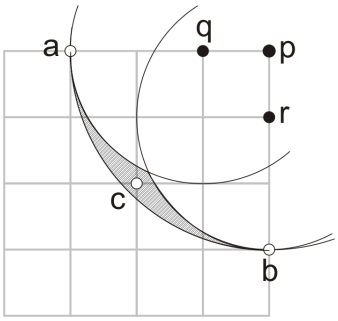

Figure 6: Pixels a, b and c are object pixels. Pixel p is the pixel under consideration, q and r are its relevant 4-neighbors.

proposes, in his ϵVDT, to store *all* nearest origins, and additionally all origins at a distance within a certain tolerance ϵ. Essentially, ϵVDT computes origin sets as defined in Equation (2). Mullikin shows that an exact distance transform is obtained when $\epsilon \geq \sqrt{D}/D$, where D is the number of spatial dimensions. This result can also be used for the other methods discussed in this paper.

The ϵVDT computes tolerance-based origin sets only as a means to compute exact DTs. Mullikin does not detail on the accuracy of the origin sets themselves in (Mullikin, 1992). Moreover, he uses only 4-neighborhoods as these are sufficient for exact Euclidean distances. We extended ϵVDT to also use 8-neighborhoods (see pseudo code in Fig. 7). These are useful for improving the origin set accuracy, as shown in Table 2. The pseudo code can be found in Figure 7. The ϵVDT is compared to the other methods in Section 8.

Besides the fact that the ϵVDT uses a scan-based approach, another conceptual difference with the FMTFT is that it uses a write formalism instead of a read formalism (Verwer et al., 1989). Whereas in the FMTFT the candidate origin set C of a pixel a is constructed by reading from all neighboring pixels (line 16 in Fig. 5), the ϵVDT writes information from a single neighbor to a (line 28 in Fig. 7).

```
 1: for each p ∈ δΩ do
 2:     S(p) ← {q ∈ δΩ | ||q − p|| ≤ ε}
 3: end for
 4: for y from 0 to N − 1 do
 5:     for x from 0 to M − 1 do
 6:         update( (x,y), (x-1,y-1) ) if s = 8
 7:         update( (x,y), (x,y-1) )
 8:         update( (x,y), (x+1, y-1) ) if s = 8
 9:         update( (x,y), (x-1, y) )
10:     end for
11:     for x from M − 1 downto 0 do
12:         update( (x,y), (x+1,y) )
13:     end for
14: end for
15: for y from N − 1 downto 0 do
16:     for x from 0 to M − 1 do
17:         update( (x,y), (x-1, y+1) ) if s = 8
18:         update( (x,y), (x, y-1) )
19:         update( (x,y), (x+1, y+1) ) if s = 8
20:         update( (x,y), (x-1, y) )
21:     end for
22:     for x from M − 1 downto 0 do
23:         update( (x,y), (x+1,y) )
24:     end for
25: end for
26: procedure update(a,b)
27: if a, b ∈ Ω then
28:     C ← S(a) ∪ S(b)
29:     D(a) ← min_{q∈C} ||q − a||
30:     S(a) ← {q ∈ C | ||q − a|| ≤ D(a) + ε}
31: end if
32: end procedure
```

Figure 7: ϵ-Vector Distance Transform (ϵVDT). The image has dimensions $M \times N$.

6 EUCLIDEAN TFT

The FMTFT visits points in the order of the inaccurate FMM distances (Sec. 4). Although this keeps the original FMM advantage of using different speed functions, an erroneous propagation order potentially influences the distance and origin set accuracy (Sec. 1). The idea comes thus naturally to design an ordered propagation FT which visits the points in order of the accurately computed distances. We present the pseudo code of this new method, called Euclidean TFT, in Figure 8. The neighborhood size s (line 15) and tolerance ϵ (line 17) have the same meaning as for the FMTFT and ϵVDT discussed above. The initialization is the same as for the FMTFT, it also uses a read formalism, and the propagation is still in the order of increasing distances. However, where the FMTFT propagates on D^{f} (Fig. 5, line 11), the ETFT propagates on the more accurate distances D (Fig. 8, line 11).

We found out that the above exact-distance propagation order yielded a comparable speed and accuracy, posing no advantage over the FMM order. Thus, we made the following change. Whereas the FMTFT updates all pixels $a \in N^{U,T}(p)$ (Fig. 5, line 13), the ETFT was made to update only pixels $a \in N^{U}(p)$ (Fig. 8, line 13). Now the ETFT updates pixels only once, trading accuracy for speed (see Table 2).

7 GRAPH-SEARCH TFT

The FMTFT and ETFT resemble the graph-search approach of (Lotufo et al., 2000). However, the graph-

```
1: for each p ∈ Ω ∪ Ω̄ do
2:     if p ∈ Ω̄ then
3:         f(p) ← K, D(p) ← −1
4:     else if p ∈ δΩ then
5:         f(p) ← T, D(p) ← 0, S(p) ← {q ∈ δΩ | ||q − p|| ≤ ε}
6:     else if p ∈ Ω ∧ p ∉ δΩ then
7:         f(p) ← U, D(p) ← ∞
8:     end if
9: end for
10: while ∃_q f(q) = T do
11:     p ← arg min_{q:f(q)=T} D(q)
12:     f(p) ← K
13:     for each a ∈ N_4^U(p) do
14:         f(a) ← T
15:         C ← ⋃_{q∈N_s^{K,T}(a)∪{a}} (S(q))
16:         D(a) ← min_{q∈C} ||q − a||
17:         S(a) ← {q ∈ C | ||q − a|| ≤ D(a) + ε}
18:     end for
19: end while
```

Figure 8: The Euclidean TFT algorithm (ETFT).

```
1: for each p ∈ Ω ∪ Ω̄ do
2:     if p ∈ Ω̄ then
3:         f(p) ← K, D(p) ← −1
4:     else if p ∈ δΩ then
5:         f(p) ← T, D(p) ← 0, S(p) ← {q ∈ δΩ | ||q − p|| ≤ ε}
6:     else if p ∈ Ω ∧ p ∉ δΩ then
7:         f(p) ← T, D(p) ← ∞
8:     end if
9: end for
10: while ∃_q f(q) = T do
11:     p ← arg min_{q:f(q)=T} D(q)
12:     f(p) ← K
13:     for each a ∈ N_s^T(p) do
14:         C ← S(a) ∪ S(p)
15:         D(a) ← min_{q∈C} ||q − a||
16:         S(a) ← {q ∈ C | ||q − a|| ≤ D(a) + ε}
17:     end for
18: end while
```

Figure 9: The Graph-search TFT algorithm (GTFT).

search method uses a write formalism, and propagates only one origin per pixel, i.e., it is a simple FT. We extended Lotufo's algorithm to a TFT, so that it can be readily compared to the FMTFT, ϵVDT, and ETFT methods. Figure 9 gives the pseudo code for this extension, called the Graph-search TFT (GTFT). Now the differences between the ETFT and GTFT methods become visible. While GTFT uses only the flags `T` and `K` and updates all neighboring pixels a of p flagged as `T` (Fig. 9, line 13), ETFT also uses the flag `U` and only updates these pixels (Fig. 8, line 13). Since there are in general less pixels flagged `U` in ETFT than `T` in GTFT, ETFT updates less pixels per iteration. However, the update of a pixel in ETFT involves more work as all neighbors of a are used (Fig. 8, line 15), whereas in GTFT only p is used (Fig. 9, line 14). The running time differences are detailed in the next section.

8 COMPARISON

Unlike DT methods, the computational complexity of (T)FT methods depends on the origin set sizes and is therefore strongly input dependent. For example, the center of a circle is a worst case, as its origin set contains all boundary points. The origin set size, for points inside convex object-regions, increases with distance to boundary. When updating a pixel p, the whole candidate origin set for p must be inspected. For N image pixels, and B boundary pixels, this poses a worst case of $O(N(B + \log B))$ for the three propagation-based methods and $O(NB)$ for ϵVDT. Luckily, average real-world images are far from this worst case. However, it is difficult to mathematically characterize the average input image. Nevertheless, to give more insight into real-world running times, we empirically compare all discussed TFT methods on speed and accuracy of both distances and origin sets. We use images that are often used as typical input for image processing algorithms.

We implemented all methods and ran them on a Pentium IV 3GHz with 1 GB RAM. Some design decisions had to be made here. For the propagation-based methods (FMTFT, ETFT, GTFT) we used a priority queue to efficiently find the temporary pixel at minimum distance to the boundary. Origin sets are stored as STL multimaps (Musser and Saini, 1996) containing (distance,origin) pairs, so that merging two origin sets takes $O(n \log n)$ time (as we must avoid duplicates), while pruning the conservative set takes $O(\log n)$. To prevent floating point precision problems when performing Statement (4), it is needed to evaluate $\|q-p\| \leq D(p)+\epsilon+\tau$ instead, where τ is larger than the minimum representable difference between two floating point numbers. However, τ must be chosen smaller than $\left|\frac{1}{2}\min_{p,q,r\in\Omega} \|\vec{pq}\| - \|\vec{pr}\|\right|$: half of the minimum difference between two distances that can occur on the grid. Alternatively, integer arithmetic can be used when the equations were rewritten. In our experiments, the use of τ improved the accuracy by two orders of magnitude.

Table 2 compares the distances D^m and origin sets S^m produced by the methods m to the exact distances D^e and origins S^e, calculated using a brute-force approach. The table shows measurements on the 'leaf' image, and cumulative measurements on 10 different images. The considered methods are the FMM, FMTFT, GTFT, ϵVDT, and ETFT. We do not con-

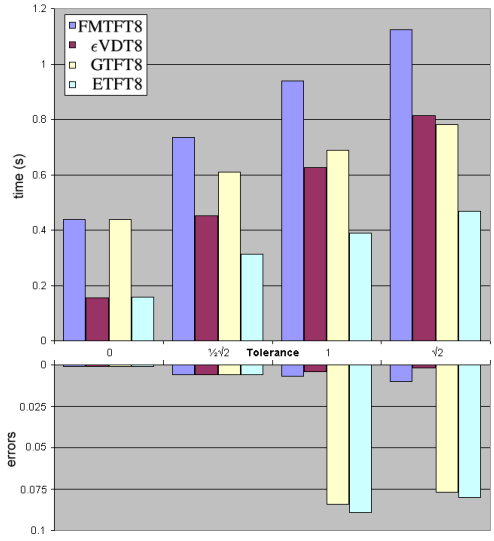

Figure 11: Timings and relative error (e_r) of the 8-neighborhood variants for the 'leaf' image.

sider the AFMM Star, as it is a simple FT. For all methods, except the FMM, we ran the 4 and 8 neighborhood variants, and used 4 different tolerances: 0, $\frac{1}{2}\sqrt{2}$, 1, and $\sqrt{2}$. For the FMM, we used only the first-order distance gradient approximation (as mentioned in Sec. 2), which needs just the 4-neighborhood. The variants are denoted as, e.g., FMTFT4 $\epsilon 0$ for the Fast Marching TFT using a 4-neighborhood and zero tolerance.

Table 2 shows that the distance errors decrease by increasing either the neighborhood size s or the tolerance ϵ. As previously noted, increasing the neighborhood size does not always eliminate all errors. Indeed, all methods produce one error for the leaf image with $s = 8$ and $\epsilon = 0$. As predicted, using $\epsilon = \frac{1}{2}\sqrt{2}$ eliminates all errors for all methods; higher tolerances are not useful for computing exact distances. Finally, our novel method ETFT4 $\epsilon\frac{1}{2}\sqrt{2}$ is the fastest of all considered methods.

We next examine the accuracy of the computed origin sets. We compare origin sets by comparing the average relative differences between a method's origins and the exact (brute-force method) origins, denoted in column '$\bar{e_r}$'. Let the relative error e_r of a pixel p be $e_r(p) = \left|\frac{|S^m(p)| - |S^e(p)|}{|S^e(p)|}\right|$, then, $\bar{e_r}$ is the average of e_r over all pixels $p \in \Omega$. The tolerance ϵ is not only a means to compute exact distances, but is also a user parameter for computing origin sets. Unlike for distances, relaxing the tolerance ϵ increases the errors for origin sets. Indeed, it is more difficult to identify all origins that are within $D(p) + \epsilon$ for higher ϵ. For $\epsilon > 0$, none of the considered methods deliver the complete origin set, although some have only a few erroneous pixels. From Table 2, we see that the 8-neighborhood variants have the best accuracy ($< 0.1\%$). Of these, ETFT8 is the fastest (see also Fig. 11). For applications needing maximum accuracy, ϵVDT8 is the method of choice. Although ϵVDT8 has a better complexity, it is probably slower because of the hidden time constant: the image is scanned twice. Finally, we illustrate the locations of the pixels with erroneous origin sets for the leaf image in Figure 10.

9 CONCLUSION

In this paper, we both analyzed and extended several distance and feature transform methods for binary images. Our goal was to provide a guide for practitioners in the field for choosing the best method that meets application-specific accuracy, speed, and output completeness criteria. First, we perfected the existing simple FT method AFMM to deliver more accurate results (AFMM Star, Sec. 3). We next extended this method to a new tolerance-based feature transform, FMTFT, that allows e.g. overcoming undesired sampling effects when computing skeletons (Sec. 4). Next, we discussed three other easy-to-implement TFT methods: the existing ϵVDT (Sec. 5), the new ETFT (Sec. 6), and the GTFT extension of Lotufo's graph-searching method (Sec. 7).

For computing exact distances, ETFT4 $\epsilon\frac{1}{2}\sqrt{2}$ is the fastest of the considered methods. Although there are other, faster, exact DT methods, e.g. (Meijster et al., 2000), the ETFT4 $\epsilon\frac{1}{2}\sqrt{2}$ can accommodate early distance-based termination and has a simple implementation (cf. Fig. 8). For computing origin sets, all methods produce fairly accurate results ($< 0.1\%$ errors) for tolerances even up to $\sqrt{2}$. ϵVDT8 is the most accurate, while ETFT8 is the fastest. Finally, FMTFT8 is still useful, as it is the only considered method that can handle different speed functions.

We next intend to extend our TFT methods to 3D and investigate their relative performance and accuracy. This should be rather straightforward, as all considered methods either do not depend on dimension (FMTFT, GTFT, ETFT) or have equivalents in higher dimensions (FMM, ϵVDT). Next, we plan to apply the TFT methods to compute robust skeletons of 3D and higher dimensional objects.

ACKNOWLEDGEMENTS

This work was supported by the Netherlands Organisation for Scientific Research (NWO) under grant number 612.065.414.

Table 2: In each row: distances D^m and origin sets S^m of method m are compared to the exact distances D^e and origins S^e. Left table: method comparison for the 'leaf' image. Right table: cumulative comparison of 10 different images. For distances (D), we show: the number of erroneous pixels ($\#e$), maximum distance error ($\max e = \max_p |D^m(p) - D^e(p)|$), and average distance error $\bar{e}$. For origins (S) we show: the number of pixels for which origin counts are different ($\#e$), and the average relative error $\bar{e_r}$ (see text). Timings are denoted in seconds in column t.

	D^m			S^m		
method m	$\#e$	$\max e$	$\bar{e}$	$\#e$	$\bar{e_r}$	t
FMM	29381	1.01	0.25	170	0.182%	0.31
FMTFT4 $\epsilon 0$	18	0.19	0.00	320	0.309%	0.31
ϵVDT4 $\epsilon 0$	22	0.19	0.00	218	0.147%	0.13
GTFT4 $\epsilon 0$	22	0.19	0.00	218	0.147%	0.28
ETFT4 $\epsilon 0$	18	0.19	0.00	317	0.304%	0.13
FMTFT8 $\epsilon 0$	1	0.04	0.00	1	0.001%	0.44
ϵVDT8 $\epsilon 0$	1	0.04	0.00	1	0.001%	0.16
GTFT8 $\epsilon 0$	1	0.04	0.00	1	0.001%	0.44
ETFT8 $\epsilon 0$	1	0.04	0.00	1	0.001%	0.16
FMTFT4 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	12428	8.452%	0.50
ϵVDT4 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	540	0.125%	0.31
GTFT4 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	31835	23.120%	0.31
ETFT4 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	26914	16.744%	0.19
FMTFT8 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	34	0.006%	0.73
ϵVDT8 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	34	0.006%	0.45
GTFT8 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	34	0.006%	0.61
ETFT8 $\epsilon \frac{1}{2}\sqrt{2}$	0	0.00	0.00	34	0.006%	0.31
FMTFT4 $\epsilon 1$	0	0.00	0.00	7674	3.879%	0.66
ϵVDT4 $\epsilon 1$	0	0.00	0.00	162	0.025%	0.42
GTFT4 $\epsilon 1$	0	0.00	0.00	22306	11.805%	0.41
ETFT4 $\epsilon 1$	0	0.00	0.00	15916	7.763%	0.25
FMTFT8 $\epsilon 1$	0	0.00	0.00	44	0.007%	0.94
ϵVDT8 $\epsilon 1$	0	0.00	0.00	32	0.004%	0.63
GTFT8 $\epsilon 1$	0	0.00	0.00	250	0.084%	0.69
ETFT8 $\epsilon 1$	0	0.00	0.00	284	0.089%	0.39
FMTFT4 $\epsilon\sqrt{2}$	0	0.00	0.00	17654	8.083%	0.75
ϵVDT4 $\epsilon\sqrt{2}$	0	0.00	0.00	142	0.018%	0.55
GTFT4 $\epsilon\sqrt{2}$	0	0.00	0.00	36718	19.525%	0.42
ETFT4 $\epsilon\sqrt{2}$	0	0.00	0.00	29446	14.286%	0.30
FMTFT8 $\epsilon\sqrt{2}$	0	0.00	0.00	43	0.010%	1.13
ϵVDT8 $\epsilon\sqrt{2}$	0	0.00	0.00	16	0.002%	0.81
GTFT8 $\epsilon\sqrt{2}$	0	0.00	0.00	273	0.077%	0.78
ETFT8 $\epsilon\sqrt{2}$	0	0.00	0.00	279	0.080%	0.47

D^m	S^m		
$\Sigma\#e$	$\Sigma\#e$	$\overline{\Sigma\bar{e_r}}$	Σt
147938	1202	0.306%	1.47
674	2792	0.576%	1.58
685	1889	0.304%	0.64
685	1891	0.304%	1.28
674	2779	0.573%	0.61
144	267	0.072%	2.09
145	217	0.054%	0.89
145	217	0.054%	2.11
144	267	0.072%	0.83
0	35828	3.888%	3.31
0	2285	0.093%	2.60
0	128672	21.655%	1.89
0	146949	17.137%	1.19
0	412	0.010%	5.16
0	392	0.009%	3.86
0	392	0.009%	3.59
0	410	0.010%	2.20
0	23854	1.835%	4.39
0	415	0.023%	3.42
0	107543	13.522%	2.53
0	89708	8.963%	1.76
0	151	0.004%	6.63
0	89	0.002%	5.24
0	1611	0.164%	4.36
0	1404	0.129%	2.81
0	59747	4.562%	5.27
0	286	0.015%	4.77
0	169396	20.611%	2.64
0	152655	14.830%	1.98
0	165	0.005%	8.39
0	27	0.000%	6.81
0	1105	0.124%	5.25
0	1130	0.157%	3.58

Figure 10: Locations of origin errors for the leaf image, $\epsilon = \sqrt{2}$. From left to right: FMTFT8, ϵVDT8, GTFT8, and ETFT8. The boundary and erroneous pixels are thickened for better display.

REFERENCES

Costa, L. and Cesar, Jr, R. (2001). *Shape analysis and classification*. CRC Press.

Cuisenaire, O. (1999). *Distance transformations: fast algorithms and applications to medical image processing*. PhD thesis, Université catholique de Louvain, Belgium.

Danielsson, P.-E. (1980). Euclidean distance mapping. *Computer Graphics and Image Processing*, 14(3):227–248.

Foskey, M., Lin, M., and Manocha, D. (2003). Efficient computation of a simplified medial axis. In *Proc. of the 8th ACM symposium on Solid modeling and applications*, pages 96–107. ACM Press.

Lotufo, T., Falcao, A., and Zampirolli, F. (2000). Fast euclidean distance transform using a graph-search algorithm. In *Proc. of the 13th Brazilian Symp. on Comp. Graph. and Image Proc.*, pages 269–275.

Meijster, A., Roerdink, J., and Hesselink, W. (2000). A general algorithm for computing distance transforms in linear time. In Goutsias, J., Vincent, L., and Bloomberg, D., editors, *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 331–340. Kluwer.

Mullikin, J. (1992). The vector distance transform in two and three dimensions. *CVGIP: Graphical Models and Image Processing*, 54(6):526–535.

Musser, D. and Saini, S. (1996). *STL tutorial and reference guide: C++ programming with the standard template library*. Addison-Wesley Professional Computing Series.

Ogniewicz, R. and Kübler, O. (1995). Hierarchic voronoi skeletons. *Pattern Recognition*, 28(3):343–359.

Ragnemalm, I. (1992). Neighborhoods for distance transformations using ordered propagation. *CVGIP: Image Understanding*, 56(3):399–409.

Sethian, J. (1999). *Level set methods and fast marching methods*. Cambridge University Press, 2nd edition.

Strzodka, R. and Telea, A. (2004). Generalized distance transforms and skeletons in graphics hardware. In *Proc. of EG/IEEE TCVG Symposium on Visualization (VisSym '04)*, pages 221–230.

Telea, A. and van Wijk, J. (2002). An augmented fast marching method for computing skeletons and centerlines. In *Proc. of the symposium on Data Visualisation*, pages 251–259.

Telea, A. and Vilanova, A. (2003). A robust level-set algorithm for centerline extraction. In *Proc. of the symposium on Data Visualisation*, pages 185–194.

Verwer, B., Verbeek, P., and Dekker, S. (1989). An efficient uniform cost algorithm applied to distance transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(4):425–429.

Ye, Q. (1988). The signed euclidean distance transform and its applications. In *Proc. of the 9th International Conference on Pattern Recognition*, volume 1, pages 495–499.