Distance transforms and feature transform sets

Wim H. Hesselink, May 4, 2009
Dept. of Mathematics and Computing Science, Rijksuniversiteit Groningen
P.O.Box 407, 9700 AK Groningen, The Netherlands
Email: `wim@cs.rug.nl`, Web: `http://www.cs.rug.nl/~wim`

Abstract

The Euclidean distance transform of a grey-scale image or volume with grey-values h is the function dt that assigns to every grid point x the minimum $h(q)+||x-q||^2$ where $||x-q||$ is the Euclidean distance between x and q. The corresponding feature transform set is the function FT that assigns to x the set of grid points q where this minimum is attained. A simple feature transform is a function ft that assigns to x some element of $FT(x)$. Extending a previous paper in IPL on binary images and volumes, we present a linear-time algorithm to compute dt, FT, and a canonical simple feature transform. Pseudocode is provided for all three functions in the 3D case.

Keywords: distance transform, feature transform, image reconstruction, image processing, computational geometry

1 Introduction

The concept of distance transformation for binary images was introduced in [14]. Given a binary image, the distance transform $dt(x)$ of a grid point x is the smallest distance of x to any point of the background. For many years, people have been satisfied with so-called chamfer distances, see [1]. The first exact Euclidean distance transform algorithms in linear time were given in [3, 2, 10]. Subsequently, other versions were invented by ourselves [12] and Maurer a.o. [11]. See [6] for a comparative survey.

Feature transform is an abbreviation of nearest feature transform [13]. There are two versions. The *Euclidean feature transform set* $FT(x)$ of a grid point x consists of the set of background grid points with minimal Euclidean distance to x. The *simple feature transform* $ft(x)$ just yields one element of $FT(x)$, say the first element in some lexical ordering of the grid points.

In [9, 8], we presented an algorithm to compute a simple feature transform in linear time, and used this to define the so-called integer medial axis (*IMA*), which is a kind of skeleton that can be computed in linear time. Indeed, for many purposes, the simple feature transform seems to be good enough. Yet, its use introduces a disturbing kind of nondeterminism.

The paper [5] uses feature transform sets (called *downstream*) to determine Euclidean skeletons of images. The algorithm of [5] to compute the (extended) downstream uses the Euclidean distance transform and a lookup table for the integral vectors of given lengths. In [7], we provided a linear-time algorithm to determine the Euclidean feature transform sets by means of a so-called concave Minarg algorithm.

We now extend the results of [7] to define and compute distance and feature transforms for grey-scale images and volumes. These grey-scale distance and feature transform algorithms are just as efficient as their binary versions. In fact, the binary versions are often aplied to grey-scale images and volumes after thresholding. Application of grey-scale versions instead may eliminate the need for thresholding.

The grey-scale distance transform proposed generalizes the so-called reverse distance transform of [15, 4], which is applied for image decoding when the image is encoded as a union of disks (balls).

Overview. In Section 2, we specify the distance transform, the simple feature transform, and the feature transform sets. In section 3, we derive the algorithm for the determination of the feature transform sets, and its derived algorithms for distance transform and the feature transform. Section 4 contains concluding remarks.

Contributions. By treating grey-scale distance and feature transforms, we include the so-called reverse distance transform of [15, 4]. The algorithm for the distance transform sets is simpler than the one of [7], and just as efficient. The derivation is also somewhat simpler. As these are only marginal improvements, we plan to publish by simple web posting.

2 Specifying distance and feature transforms

We first set the stage for grey-scale images in arbitrary dimension d (though usually $d = 2$ or 3). The grid $\mathbb{Z}^d$ is regarded as a subset of the Euclidean vector space $\mathbb{R}^d$ with its standard length function. For grid points x and y, the squared distance is $||x - y||^2 = \sum_i (x_i - y_i)^2$. This is always a natural number. Computation of the square root is usually superfluous.

We define an *image* to be a pair (A, h) where A is a rectangular box in $\mathbb{Z}^d$ and $h : A \to \mathbb{R}$ is a grey level function (integer or float). The grey-level function h is combined with the Euclidean distance in A to a function $f : A^2 \to \mathbb{R}$ given by $f(x, p) = ||p - x||^2 + h(p)$. We then define

$$dt(x, h) = \text{Min}\{f(x, p) \mid p \in A\} \ ,$$
$$FT(x, h) = \{p \in A \mid f(x, p) = dt(x, h)\} \ .$$

The reader may raise the objection that adding grey-values to squared distances is like comparing apples to pears. Indeed, in general, we propose to apply the algorithm with $h(p)$ not the grey-value itself, but some monotonic or antimonotonic expression in the grey-value, which should depend very much on the application. In applications where distances should compensate grey-values linearly, one could take $h(p)$ proportional to the square of the grey-value.

If we assume that $h(p)$ ranges from 0 (black) to *white*, one criterion for scaling could be the requirement that a single dark spot on a white background should have impact everywhere in box A and in particular at the opposite corner. This may suggest the condition $diag^2 \leq white$, where *diag* is the length of the diagonal of the box.

2.1 Two applications to binary images

A *binary image* is a pair (A, B) where A is a rectangular box of grid points in $\mathbb{Z}^d$ as before and B is a subset of A called the *background*. The complement $Fg = A \setminus B$ of B within A is called the *foreground*.

The *canonical* distance and feature transforms of (A, B) are obtained by taking some upper bound $W \geq ||x - p||$ for all pairs $x, p \in A$. We then define

$$\begin{aligned} &dt(x, B) = dt(x, h_B) \ , \\ &FT(x, B) = FT(x, h_B) \ , \\ &\text{where} \ \ h_B(y) = (y \in B \ ? \ 0 : W) \ . \end{aligned}$$

W stands for *white*. If B is nonempty, $dt(x, B)$ and $FT(x, B)$ are independent of the choice of W because, for every x, there is some background point with square distance $\leq W$ to x. In that case, $dt(x, B)$ and $FT(x, B)$ are the same as defined elsewhere, e.g., [9].

We also consider encodings of a binary image Fg, as a union of balls with a set C of centers and square radii $r(c)$ for $c \in C$. Formally, we define a pair (C, r) where $C \subseteq A$ and $r : C \to \mathbb{N}$, to be a *covering pair* if and only if

$$Fg = \{x \in A \mid \exists \, c \in C : ||x - c||^2 < r(c)\} \ .$$

When C can be taken small, this is an efficient encoding in the sense that it does not take much space to store the pair (C, r). This raises two questions.

The first question is: how to reconstruct Fg or B efficiently from a covering pair (C, r)? This question is reduced to the grey-scale distance transform by observing that we have:

$$\begin{aligned} Fg &= \{x \in A \mid \text{Min}\{||x - c||^2 - r(c) \mid c \in C\} < 0\} \\ &= \{x \in A \mid dt(x, h_r) < 0\} \ , \\ &\text{where} \ \ h_r(y) = (y \in C \ ? \ -r(y) : 0) \ . \end{aligned}$$

This variation of the Euclidean distance transform is called the *reverse distance transform* in [15, 4].

The second question is how to compute a small covering pair (C, r) efficiently for a given set B? We hope to treat this question elsewhere.

3 Computation of distance and feature transforms

In this section we develop an algorithm for the feature transform sets $FT(x, h)$ for all $x \in A$. The basic idea is to use induction in the dimension. It turns out that the base case is more complicated than the induction step.

Indeed, the one-dimensional case is treated in the sections 3.1, 3.2, and 3.3. In Section 3.4, we treat the cases with dimension > 1, inductively. Section 3.5 presents pseudocode for the determination of the feature transform sets in the 3D case.

When h is fixed, we write $dt(x)$ and $FT(x)$ instead of $dt(x, h)$ and $FT(x, h)$. A *simple feature transform* is a function ft with $ft(x) \in FT(x)$ for all x.

3.1 The one-dimensional case

The base case has dimension $d = 1$. In this case, A is a grid line and we may assume that it consists of the integers x with $0 \leq x < n$ for fixed n. So, $A = [0 \ldots n) \subseteq \mathbb{N}$ and we have to develop an algorithm to determine $FT(x, h)$ for all $x \in [0 \ldots n)$. In this case, function f of section 2 satisfies $f(x, p) = (p - x)^2 + h(p)$ for all x and $p \in A$.

For the development of the algorithm, we keep the range $[0 \ldots n)$ of x constant but replace the range of p by $[0 \ldots k)$ where k is a variable. Now $FT(x, h) = M(x, n)$ where $M(x, k)$ is defined as the set of numbers $p \in [0 \ldots k)$ for which $f(x, p)$ is minimal. So, we have

$$(0) \qquad M(x, k) = \{p \in [0 \ldots k) \mid \forall\, q \in [0 \ldots k) : f(x, p) \leq f(x, q)\} \ .$$

For every p, the graph of function $f(\_, p)$ is a parabola with top in $(p, h(p))$. Therefore, $M(x, k)$ consists of the indices $< k$ of the parabolas that are minimal at the argument x.

We obviously have $\emptyset \neq M(x, k) \subseteq [0 \ldots k)$ for all $k \geq 1$ and all x. In particular, $M(x, 1) = \{0\}$. With respect to incrementation of k, the definition of function M implies that $M(x, k + 1) = M(x, k)$ if $f(x, k)$ is bigger than the minimal value of $f(x, \_)$ on $[0 \ldots k)$, that $M(x, k + 1) = \{k\}$ if $f(x, k)$ is less than this minimal value, and that $M(x, k + 1) = M(x, k) \cup \{k\}$ in the case of equality. This shows that, for every $p \in M(x, k)$, we have the conditional expression

$$\begin{array}{ll} (1) & M(x, k + 1) \quad = \\ & \quad (\ f(x, p) < f(x, k)\ ?\ M(x, k) \\ & \quad :\ f(x, p) = f(x, k)\ ?\ M(x, k) \cup \{k\} :\ \{k\}\ )\ . \end{array}$$

This recurrence relation leads to a straightforward algorithm for the computation of the set $M(x, n)$, which is of order $\mathcal{O}(n)$. When we need to apply this for all $x \in [0 \ldots n)$, we get an algorithm of order $\mathcal{O}(n^2)$.

To get a more efficient algorithm, we use that function M is in some sense monotonic in its first argument:

$$(2) \qquad x < y \quad \wedge \quad p \in M(x, k) \quad \wedge \quad q \in M(y, k) \quad \Rightarrow \quad p \leq q\ .$$

This is proved in

$$\begin{array}{ll} & x < y \quad \wedge \quad p \in M(x, k) \quad \wedge \quad q \in M(y, k) \\ \Rightarrow & \quad \{(0) \text{ twice } \} \\ & x < y \quad \wedge \quad f(x, p) \leq f(x, q) \quad \wedge \quad f(y, q) \leq f(y, p) \\ \Rightarrow & \quad \{ \text{ addition } \} \\ & x < y \quad \wedge \quad f(x, p) + f(y, q) \leq f(x, q) + f(y, p) \\ \equiv & \quad \{ \text{ definition of } f \text{ and cancellation of } h \text{ terms } \} \\ & x < y \quad \wedge \quad (p - x)^2 + (q - y)^2 \leq (q - x)^2 + (p - y)^2 \\ \equiv & \quad \{ \text{ the squares } p^2,\ q^2,\ x^2,\ y^2 \text{ cancel } \} \\ & 0 < y - x \quad \wedge \quad 0 \leq 2 \cdot (q - p) \cdot (y - x) \\ \Rightarrow & \quad \{ \text{ calculus } \} \\ & p \leq q\ . \end{array}$$

For $k > 0$ and $x \in [0 \ldots n)$, we define $a(x,k)$ to be the minimum of $M(x,k)$. We have $0 \le a(x,k) \le k-1$. It is convenient also to define $a(n,k) = k-1$. Formula (2) with $y := x+1$ now implies that for all $x \in [0 \ldots n)$:

$$a(x,k) \le a(x+1,k) \ . \tag{3}$$

It even implies $p \le a(x+1,k)$ for every $p \in M(x,k)$. Consequently, we get for all $x \in [0 \ldots n)$ and $0 < k \le n$:

$$M(x,k) = \{p \mid a(x,k) \le p \le a(x+1,k) \ \wedge \ f(x,p) = f(x,a(x,k))\} \ . \tag{4}$$

We now aim at an inductive expression for $a(\_,k+1)$ in $a(\_,k)$. The set of arguments x with $a(x,k+1) = k$ is a nonempty final segment of $[0 \ldots n]$. We therefore define the function u as the starting point of this final segment:

$$u(k) = \min \{x \mid a(x,k+1) = k\} \ .$$

For $x \in [0 \ldots n)$, we have

$$\begin{array}{ll} & x < u(k) \\ \equiv & a(x,k+1) < k \\ \equiv & \{ \ (0) \text{ with } p := a(x,k+1) \text{ and } q := k \ \} \\ & f(x,a(x,k)) \le f(x,k) \ . \end{array} \tag{5}$$

Since $a(x,k) \in M(x,k)$, it follows from (5), that formula (1) implies

$$a(x,k+1) \quad = \quad (\ x < u(k) \ ?\ a(x,k) :\ k\ ) \ . \tag{6}$$

In order to apply formula (5), we observe that

$$\begin{array}{ll} & f(x,p) \le f(x,q) \\ \equiv & (p-x)^2 + h(p) \le (q-x)^2 + h(q) \\ \equiv & 2x(q-p) \le q^2 - p^2 + h(q) - h(p) \\ \equiv & \{ \text{ assume } p < q \ \} \\ & x \le (q^2 - p^2 + h(q) - h(p))/(2(q-p)) \ . \end{array}$$

We therefore define the *separator function* g by

$$g(p,q) = \lfloor (q^2 - p^2 + h(q) - h(p))/(2(q-p)) \rfloor \ .$$

If $p < q$, this function g satisfies for all integer x:

$$f(x,p) \le f(x,q) \quad \equiv \quad x \le g(p,q) \ . \tag{7}$$

Notice that $g(p,q)$ is not bounded to $[0 \ldots n)$.

Formulas (5) and (7) determine $u(k)$ in terms of $a(\_,k)$, while formula (6) determines $a(\_,k+1)$ in terms of $u(k)$ and $a(\_,k)$. Therefore, the formulas together are enough for inductive computation of $a(\_,n)$.

In the corresponding section of [7], we also had a recurrence relation expressing $M(x,k+1)$ in terms of $M(x,k)$. The use of this relation is eliminated here. In fact, we shall see in section 3.3 that $M(\_,n)$ can be recovered from $a(\_,n)$ in linear time.

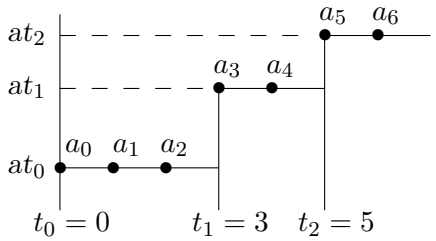
3.2 Algorithm design

By formula (6), function $a(\_, k+1)$ is obtained from $a(\_, k)$ by modification at the end by means of a constant function. It is therefore algorithmically convenient to store $a(\_, k)$ as a step function.

Since the sequence $a(\_, k)$ is ascending by (3), we are especially interested in the indices where it increases and in the corresponding values. Let $Q(k)$ be the set of indices $x \in [1 \ldots n)$ with $a(x-1, k) < a(x, k)$. Put $q(k) = \#Q(k)$ and let $t(1, k)$ up to $t(q(k), k)$ be be the increasing enumeration of the elements of $Q(k)$. We also define $t(0, k) = 0$ and $t(q(k)+1, k) = n$ and $at(i, k) = a(t(i, k), k)$ for $0 \le i \le q(t)$. Then we have

$$
\begin{array}{lll}
0 \le i \le q(k) & \Rightarrow & t(i,k) < t(i+1,k) \ , \\
0 \le i < q(k) & \Rightarrow & at(i,k) < at(i+1,k) \ , \\
0 \le i \le q(k) \ \wedge \ t(i,k) \le x < t(i+1,k) & \Rightarrow & a(x,k) = at(i,k) \ .
\end{array} \tag{8}
$$

In the next diagram, this is illustrated for fixed k using the abbreviations $a_x = a(x, k)$, $t_i = t(i, k)$, and $at_i = at(i, k)$ for the case that $t_1 = 3$ and $t_2 = 5$.



The base case is $k = 1$. Then we have $q(1) = 0$, and $at(0, 1) = 0$.

Assume that, for given $k \ge 1$, we know $q(k)$ and the numbers $t(i, k)$, and $at(i, k)$ for $i \le q(k)$. Formula (5) implies for $i \le q(k)$

$$
t(i,k) < u(k) \quad \equiv \quad f(t(i,k), at(i,k)) \le f(t(i,k), k) \ . \tag{9}
$$

Note that $0 \le u(k) \le n$. Since $t(i, k)$ is increasing in i, we can use linear search with formula (9) to determine whether $u(k) = 0$ or to determine the greatest index $j \le q(t)$ with $t(j, k) < u(k)$.

If $u(k) = 0$, formula (6) implies $q(k+1) = 0$ and $at(0, k+1) = k$.

It remains to consider $0 < u(k) \le n$, say $t(j, k) < u(k) \le t(j+1, k)$. For any x with $t(j, k) \le x < t(j+1, k)$, we observe

$$
\begin{array}{lll}
 & x < u(k) & \\
\equiv & \{ \ (5) \ \} & f(x, a(x,k)) \le f(x,k) \\
\equiv & \{ \ (8) \ \} & f(x, at(j,k)) \le f(x,k) \\
\equiv & \{ \ (7) \ \} & x \le g(at(j,k), k) \ .
\end{array}
$$

Substituting $x := u(k) - 1$, this yields

$$
u(k) \le 1 + g(at(j,k), k) \ .
$$

To get the reverse inequality, we observe

$$
\begin{array}{ll}
 & u(k) < n \\
\Rightarrow & \{\ (5) \text{ with } x := u(k)\ \} \\
 & f(u(k),k) < f(u(k), a(u(k),k)) \\
\Rightarrow & \{\ \text{definition } M \text{ and } a\ \} \\
 & \forall\, q \in [0 \ldots k) : f(u(k),q) > f(u(k),k) \\
\equiv & \{\ (7) \text{ and } q < k\ \} \\
 & \forall\, q \in [0 \ldots k) : g(q,k) < u(k) \\
\Rightarrow & \{\ q := at(j,k) \in [0 \ldots k)\ \} \\
 & 1 + g(at(j,k),k) \leq u(k)\ .
\end{array}
$$

In any case, we have $u(k) \leq n$ by definition. We thus get

$$u(k) = \min(n, 1 + g(at(j,k),k))\ .$$

Now that we have determined $u(k)$, we can proceed to determine $q(k+1)$ and the numbers $t(i,k+1)$ and $at(i,k+1)$ for $i \leq q(k+1)$.

For $x < u(k)$, it follows from (6) that $a(x,k+1) = a(x,k)$, so that the data remain unchanged. On the other hand, for $u(k) \leq x < n$, we have $a(x,k+1) = k$. Therefore $u(k)$ is the last index in the sequence $t(\_,k+1)$ before n and we have $a(u(k),k+1) = k$. This concludes the algorithmic analysis.

3.3 Data representation

It remains to encode the algorithm presented above in such a way that we can efficiently gather the sets $M(x,n)$ for all x.

We first declare program variables to hold the data in (8). We let `k`, which represents k, proceed from 1 to n in fragment *Build*. Fragment *FTharvest* collects and returns an array `FT` that satisfies $\texttt{FT}(x) = M(x,n)$ for all y.

oneFT$(n : \mathbb{N};\ h : [0 \ldots n) \to \mathbb{R}) =$
  var $\texttt{k}, \texttt{q} : \mathbb{Z}\ ;\ \texttt{t}, \texttt{at} : [0 \ldots n] \to \mathbb{Z}\ ;$
  Build ;
  FTharvest .

In both *Build* and *FTharvest*, we use the functions f and g as they are defined above in terms of h.

We let `q` stand for $q(\texttt{k})$ and, similarly, preserve the invariants $\texttt{t}(i) = t(i,\texttt{k})$ and $\texttt{at}(i) = at(i,\texttt{k})$ for all $0 \leq i \leq \texttt{q}$. The analysis in 3.1 and 3.2 leads to the following algorithm that preserves these invariants:

Build :
  $\texttt{q} := 0\ ;\quad \texttt{t}(0) := 0\ ;\quad \texttt{at}(0) := 0\ ;$
  for $\texttt{k} := 1$ **to** $n-1$ **do**
    LinearSearch ;
    if $\texttt{q} < 0$ **then** *Reset*
    else *Update* **end**
  end .

The fragment *LinearSearch* makes $\texttt{q} := j$ where j is maximal with $j < 0$ or $t(j,k) < u(k)$:

LinearSearch :
while $\mathtt{q} \geq 0 \wedge f(\mathtt{t}(\mathtt{q}), \mathtt{at}(\mathtt{q})) > f(\mathtt{t}(\mathtt{q}), \mathtt{k})$ **do** $\mathtt{q} := \mathtt{q} - 1$ **end** .

Fragment *Reset* handles the case $u(k) = 0$:

Reset :
$\mathtt{q} := 0$; $\mathtt{at}(0) := \mathtt{k}$.

Fragment *Update* determines $u(k)$ and then restores the invariants:

Update :
$\mathtt{w} := 1 + g(\mathtt{at}(\mathtt{q}), \mathtt{k})$;
if $\mathtt{w} < n$ **then**
$\mathtt{q} := \mathtt{q} + 1$;
$\mathtt{t}(\mathtt{q}) := \mathtt{w}$;
$\mathtt{at}(\mathtt{q}) := \mathtt{k}$;
end .

We use $vf = 2(n - \mathtt{k}) + \mathtt{q}$ to analyse the complexity of the algorithm *Build*. Initially $vf = 2n - 2$. Since vf decreases both in the body of *LinearSearch* and in the body of the outer loop, *Build* has time complexity linear in n.

It remains to determine the sets $\mathtt{FT}(x) = M(x, n)$ by means of formula (4). This is implemented in

FTharvest :
var $\mathtt{FT} : [0 \ldots n) \to \mathbb{P}(\mathbb{Z})$;
$\mathtt{t}(\mathtt{q}+1) := n$; $\mathtt{at}(\mathtt{q}+1) := n - 1$;
for $j := 0$ **to** $\mathtt{q}$ **do**
$x1 := t(j+1) - 1$; // used five times
for $x := \mathtt{t}(j)$ **to** $x1$ **do** $\mathtt{FT}(x) := \{\mathtt{at}(j)\}$ **end** ;
for $p := \mathtt{at}(j) + 1$ **to** $\mathtt{at}(j+1)$ **do**
if $f(x1, p) = f(x1, \mathtt{at}(j))$ **then**
$\mathtt{FT}(x1) := \mathtt{FT}(x1) \cup \{p\}$ **end end end** ;
return FT .

The time complexity of this collection phase is of the order of

$$\textstyle\sum_{j=0}^{q}(\mathtt{t}(j+1) - \mathtt{t}(j) + 1 + \mathtt{at}(j+1) - \mathtt{at}(j)) \leq 3n \ .$$

Therefore the complete algorithm is of order linear in n.

The main difference with the algorithm of [7] is that all set manipulations are moved from *Build* and *Update* to *FTharvest*. This variation has the advantage that the scaled-down versions can be easily described.

For example, if one only wants the simple feature transform, one just replaces in *oneFT* the fragment *FTharvest* by:

ftHarvest :
var $\mathtt{ft} : [0 \ldots n) \to \mathbb{Z}$;
$\mathtt{t}(\mathtt{q}+1) := n$;
for $j := 0$ **to** $\mathtt{q}$ **do**
for $x := \mathtt{t}(j)$ **to** $t(j+1) - 1$ **do**
$\mathtt{ft}(x) := \mathtt{at}(j)$ **end end** ;
return ft .

If one only needs the distance transform itself, one replaces *FTharvest* by:

```
dtHarvest :
    var  dt : [0 . . . n) → ℝ ;
    t(q + 1) := n ;
    for  j := 0  to  q  do
        for  x := t(j)  to  t(j + 1) − 1  do
            dt(x) := f(x, at(j))  end end ;
    return dt .
```

These are the algorithm described in [8, 9, 12].

Note that the specific form of function f only enters in the formulas (2) and (7). Formula (2) is essential. If a function f satisfies (2) but there is no function g with (7), the application of (7) in *Update* can be replaced by a binary search.

3.4 Higher dimensions

As announced, the higher dimensional cases are treated by induction in the dimension. It turns out that the base case also serves in the step.

In the induction step, we assume that $d > 1$ and that algorithms to compute dt and FT in dimension $d-1$ are available. We then have to compute dt and FT in dimension d. The rectangular box A is now a cartesian product of the form $A = A' \times [0 \ldots n)$ where A' is a rectangular box in $\mathbb{Z}^{d-1}$ and $n \in \mathbb{N}$. Function h is now $A' \times [0 \ldots n) \to \mathbb{R}$.

For every $y \in A'$, we shall compute $FT(x, h)$ for all grid points $x = (y, z)$ with $0 \le z < n$. So, we let $y \in A'$ be fixed. By the Theorem of Pythagoras, the distance transform $dt(x, h)$ of a grid point $x = (y, z)$ is the minimum value of $(q - z)^2 + h(q)$, where q ranges over $[0 \ldots n)$ and $h(q) = dt(y, B_q)$. In formulas, this looks like:

$$\begin{aligned} & dt((y,z),h) \\ = \;& \mathrm{Min}\{||(p,q)-(y,z)||^2 + h(p,q) \mid q \in [0\ldots n),\ p \in A'\} \\ = \;& \mathrm{Min}\{(q-z)^2 + \mathrm{Min}\{||p-y||^2 + h(p,q) \mid p \in A'\} \mid q \in [0\ldots n)\} \\ = \;& \mathrm{Min}\{(q-z)^2 + h'(q) \mid q \in [0\ldots n)\}\ , \end{aligned}$$

where $h'(q) = \mathrm{Min}\{||p - y||^2 + h(p, q) \mid p \in A'\} = dt(y, h(\_, q))$. Now, $h'(q)$ can be computed since it is a lower dimensional distance transform. Similarly, the feature transform $FT((y, z), h)$ of (y, z) is the set of pairs (p, q) with $q \in FT(z, h')$ and $p \in FT(y, h(\_, q))\}$. We thus obtain

$$\begin{aligned} & dt_d((y,z),h) = dt_1(z,h'), \\ & FT_d((y,z),h) = \{(p,q) \mid q \in FT_1(z,h'),\ p \in FT_{d-1}(y,h(\_,q))\} \\ & \quad \text{where } h' = (\lambda q : dt_{d-1}(y,h(\_,q)). \end{aligned}$$

Here, we have given the functions dt and FT indices to indicate the dimensions. This shows that the d-dimensional versions are expressed in a lower-dimensional version and a one-dimensional version.

The occurrence of dt in the recurrence relation for FT can be eliminated by the observation that $h'(q) = dt_{d-1}(y, h(\_, q)) = ||p - y||^2 + h(p, q)$ for any $p \in FT_{d-1}(y, h(\_, q))$.

3.5 The 3D case

We now only treat the case of a 3D box A with dimensions xL, yL, zL. So box A is defined by the conditions $0 \leq x < xL$, $0 \leq y < yL$, $0 \leq z < zL$.

The algorithm has three phases. In the first phase, *FT1* is constructed. For every $(x, y, z) \in A$, the set $FT1(y, z, x)$ holds the numbers p such that (p, y, z) has minimal $(p - x)^2 + h(p, y, z)$. In the second phase, *FT2* is constructed. For every $(x, y, z) \in A$, the set $FT2(z, x, y)$ holds the pairs (p, q) such that (p, q, z) has minimal $||(p, q) - (x, y)||^2 + h(p, q, z)$. In the third phase, the actual distance transform sets FT are constructed.

$$
\begin{array}{l}
\mathit{threeFT}(xL, yL, zL : \mathbb{Z};\ h : A \to \mathbb{R}) = \\
\quad \textbf{var}\ \ \mathtt{FT1} : A_1 \to \mathbb{P}(\mathbb{Z})\ ;\ \ \mathtt{FT2} : A_2 \to \mathbb{P}(\mathbb{Z}^2)\ ;\ \ \mathtt{FT} : A \to \mathbb{P}(\mathbb{Z}^3)\ ; \\
\qquad \mathtt{hh} : \mathbb{Z} \to \mathbb{Z}\ ;\ \ \mathtt{mm} : \mathbb{Z} \to \mathbb{P}(\mathbb{Z})\ ; \\
\quad \textbf{for all}\ y, z\ \textbf{do}\ \ \mathtt{FT1}(y, z) := \mathit{oneFT}(xL, h(\_, y, z))\ \ \textbf{end}; \\
\quad \textbf{for all}\ z, x\ \textbf{do} \\
\qquad \mathtt{hh} := \lambda q : (p - x)^2 + h(p, q, z)\ \mathit{with}\ p \in \mathtt{FT1}(q, z, x)\ ; \\
\qquad \mathtt{mm} := \mathit{oneFT}(yL, \mathtt{hh})\ ; \\
\qquad \textbf{for all}\ y\ \textbf{do} \\
\qquad\quad \mathtt{FT2}(z, x, y) := \{(p, q) \mid q \in \mathtt{mm}(y),\ p \in \mathtt{FT1}(q, z, x)\} \\
\qquad \textbf{end} \\
\quad \textbf{end}\ ; \\
\quad \textbf{for all}\ x, y\ \textbf{do} \\
\qquad \mathtt{hh} := \lambda r : ||(p, q) - (x, y)||^2 + h(p, q, r) \\
\qquad\qquad\qquad \mathit{with}\ (p, q) \in \mathtt{FT2}(r, x, y)\ ; \\
\qquad \mathtt{mm} := \mathit{oneFT}(zL, \mathtt{hh})\ ; \\
\qquad \textbf{for all}\ z\ \textbf{do} \\
\qquad\quad \mathtt{FT}(x, y, z) := \{(p, q, r) \mid r \in \mathtt{mm}(z),\ (p, q) \in \mathtt{FT2}(r, x, y)\} \\
\qquad \textbf{end} \\
\quad \textbf{end}\ ; \\
\quad \textbf{return}\ \mathtt{FT}\ .
\end{array}
$$

The set A_1 consists of the triples (y, z, x) with $(x, y, z) \in A$. Similarly, A_2 consists of the triples (z, x, y) with $(x, y, z) \in A$.

Note that the choices made by choosing elements of `FT1` and `FT2` are irrelevant for the result because all elements of a feature transform set have the same distance. Alternatively, one can extend method *oneFT* in such a way that it also yields the corresponding distance transform. In that case, the computations of `hh` can be simplified. We rejected this solution because of its memory requirements.

Remark. For most purposes, one would prefer the scaled-down version of the algorithm that yields the simple feature transform *ft*, which is characterized by the fact that $ft(x, y, z)$ is the lexically first element of $FT(x, y, z)$. The scaled-down version is obtained by scaling-down *oneFT* in such a manner that it yields arrays $\mathtt{ft} : \mathbb{Z} \to \mathbb{Z}$. For this purpose, fragment *FTharvest* is replaced by *ftHarvest*. In *threeFT*, the arrays `FT1`, `FT2`, and `FT` are replaced by

$$\mathtt{ft1} : A_1 \to \mathbb{Z}\ ;\ \ \mathtt{ft2} : A_2 \to \mathbb{Z}^2\ ;\ \ \mathtt{ft} : A \to \mathbb{Z}^3\ .$$

The choices in the definitions of `hh` are removed and the assignments to `FT2` and `FT` are replaced by

$$\texttt{ft2}(z,x,y) := (\texttt{ft1}(q,z,x),q) \quad \text{where} \quad q = \texttt{mm}(y)\ ;$$
$$\texttt{ft}(x,y,z) := (\texttt{ft2}(r,z,x),r) \quad \text{where} \quad r = \texttt{mm}(z)\ .$$

So the main difference is the removal of the second inner for-loop of *Harvest* and the elimination of the set-building operations. □

4 Concluding Remarks

The present algorithm is a minor modification of the one in [7], and actually closer to the one of [12].

We compared a prototype implementation in Java with a similar one for the program of [7]. Both versions now analyse the blood vessel volume of $128 \times 128 \times 62$ voxels in 1.35 seconds on a Pentium 4 (3GHz). The deviations are such that no preference can be found.

References

[1] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, graphics, and Image Processing*, 27:321–345, 1984.

[2] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time Euclidean distance transform algorithms. *IEEE Trans. Pattern Anal. Machine Intell.*, 17:529–533, 1995.

[3] L. Chen and H.Y.H. Chuang. A fast algorithm for Euclidean distance maps of a 2-d binary image. *Inf. Process. Lett.*, 51:25–29, 1994.

[4] D. Coeurjolly. d-Dimensional reverse Euclidean distance transformation and Euclidean medial axis extraction in optimal time. In Nyström et al., editor, *Discrete Geometry for Computer Imagery 2003*, volume 2886 of *LNCS*, pages 327–337, New York, 2003. Springer.

[5] M. Couprie and R. Zrour. Discrete bisector function and Euclidean skeleton. In E. Andres, G. Damiand, and P. Lienhardt, editors, *Discrete Geometry for Computer Imagery 2005*, volume 3429 of *LNCS*, pages 216–227. Springer, 2005.

[6] R. Fabbri, L.D.F Costa, J.C. Torelli, and O.M. Bruno. 2D Euclidean distance transform algorithms: a comparative survey. *ACM Comput. Surv.*, 40(1–2), 2008.

[7] W.H. Hesselink. A linear-time algorithm for Euclidean feature transform sets. *Inf. Process. Lett.*, 102:181–186, 2007.

[8] W.H. Hesselink and J.B.T.M. Roerdink. Euclidean skeletons of image and volume data in linear time by integer medial axis transform. *IEEE Trans. Pattern Anal. Machine Intell.*, 30:2204–2217, 2008.

[9] W.H. Hesselink, M. Visser, and J.B.T.M. Roerdink. Euclidean skeletons of 3D data sets in linear time by the integer medial axis transform. In C. Ronse, L. Najman, and E. Decencière, editors, *Mathematical Morphology: 40 Years On (Proc. 7th Intern. Symp. on Mathematical Morphology, April 18-20)*, pages 259–268. Springer V., 2005.

[10] T. Hirata. A unified linear-time algorithm for computing distance maps. *Inf. Process. Lett.*, 58:129–133, 1996.

[11] C.R. Maurer Jr., R. Qi, and V. Raghavan. A linear time algorithm for computing the Euclidean distance transform in arbitrary dimensions. *IEEE Trans. Pattern Anal. Machine Intell.*, 25:265–270, 2003.

[12] A. Meijster, J.B.T.M. Roerdink, and W.H. Hesselink. A general algorithm for computing distance transforms in linear time. In J. Goutsias, L. Vincent, and D.S. Bloomberg, editors, *Mathematical morphology and its applications to image and signal processing (Proc. 5th Int. Conf.)*, pages 331–340. Kluwer, 2000.

[13] D.W. Paglieroni. A unified distance transform algorithm and architecture. *Machine Vision and Applications*, 5:47–55, 1992.

[14] A. Rosenfeld and J.L. Pfaltz. Sequential operations in digital picture processing. *Journal ACM*, 13:471–494, 1966.

[15] T. Saito and J.-I. Toriwaki. Reverse distance transformation and skeletons based upon the Euclidean metric for n-dimensional digital pictures. *IECE Trans. Inf. & Syst.*, E77-D(9):1005–1016, 1994.