A Generalized Squared Euclidean Distance Transform with Voronoi Maps

B. King, R. Döker, S. Meier, H. Shin, and M. Galanski

May 22, 2006

Department of Diagnostic Radiology
Hannover Medical School
Carl-Neuberg-Str. 1
D-30625 Hannover
Germany

Abstract

This document describes the implementation of an algorithm that computes a generalization of the distance transform with the squared euclidean metric.

The generalization allows for interesting image operators, e.g. a morphologic dilation with euclidean ball structure elements that can vary in size across the image. Voronoi maps and the standard distance transform can be computed as well.

The algorithm is provided as an image processing filter for ITK. Several example programs demonstrate its applications.

Contents

1 Introduction

Distance transforms are versatile image processing operators rich in applications. They are used in level set segmentation, watershed segmentation, skeletonization, and medial axis transform, for example [Sethian, 1996, Roerdink and Meijster, 2000, Toriwaki and Mori, 2001, Ge and Fitzpatrick, 1996]. They can also be applied to compute dilations and erosions for spherical structure elements in mathematical morphology [Cuisenaire, 2006].

Since the seminal paper by Danielsson [1980], the canonical solution to efficient distance transforms is the algorithm described therein. ITK provides an N-dimensional implementation that computes a Voronoi map and a vector distance map as well.

There are quite a few algorithms for computing a distance transform some of which are in the same complexity class as the one by Danielsson while also achieving better precision. Cuisenaire [1999] gives an excellent survey in his PhD thesis. Recently, a fast algorithm by Maurer et al. [2003] has been implemented in ITK by Tustison et al. [2006].

In search of an algorithm that can compute the union of many spheres efficiently, we discovered a generalization of the distance transform which fits to that problem rather nicely [Felzenszwalb and Huttenlocher, 2004]. The algorithm has linear complexity in the number of voxels in the image: $O(NM^N)$ for an N-dimensional image with M^N voxels. Felzenszwalb and Huttenlocher [2004] provide an implementation in C.

We implemented this algorithm as an `itk::ImageToImageFilter` and added the possibility to generate Voronoi maps.

The remainder of this paper is structured as follows: First, we will define the generalized distance transform and explain the algorithm. Next, we will present our implementation and subsequently give examples for its use. This is followed by an analysis of runtime and memory performance and by closing remarks.

In addition to the image filter itself, we distribute all other code and data that took part in creation of this paper.

2 Generalized Distance Transforms of Sampled Functions

Given an image domain $D \subset \mathbb{R}^N$, a segmented object $S \subset D$, and a metric $d : D \times D \to \mathbb{R}$, the *distance transform* DT_S of S is defined as:

$$\mathrm{DT}_S : \begin{cases} D & \to \mathbb{R} \\ x & \mapsto \min_{s \in S}(d(x,s)) \end{cases} \tag{1}$$

An alternative is to provide S in form of an *indicator function* $i_S : D \to \{0, \infty\}$:

$$i_S(x) := \begin{cases} 0 & \text{if } x \in S, \\ \infty & \text{otherwise.} \end{cases} \tag{2}$$

The distance transform of such a function is then given by:

$$\mathrm{DT}_{i_S}(x) := \min_{y \in D}(d(x,y) + i_S(y)) \tag{3}$$

Felzenszwalb and Huttenlocher [2004] generalize this concept by allowing arbitrary functions $f : D \to \mathbb{R} \cup \{\infty\}$ in place of i_S.

2.1 Squared Euclidean Distance

One algorithm described in [Felzenszwalb and Huttenlocher, 2004] uses the squared euclidean distance metric $d(x,y) = (x-y)^2$. It is of importance, because in multidimensional images, it allows for an iterative solution where each step is one-dimensional. The following example is formulated for $D = \mathbb{R}^2$ but extends to arbitrary dimension:

$$\begin{aligned} & \mathrm{DT}_f((x_1,x_2)) \\ = & \min_{(s_1,s_2)\in\mathbb{R}^2}((s_1-x_1)^2 + (s_2-x_2)^2 + f(s_1,s_2)) \\ = & \min_{s_2\in\mathbb{R}}((s_2-x_2)^2 + \min_{s_1\in\mathbb{R}}((s_1-x_1)^2 + f(s_1,s_2))) \\ = & \min_{s_2\in\mathbb{R}}((s_2-x_2)^2 + \mathrm{DT}_{f|_{s_2}}(x_1)) \end{aligned} \tag{4}$$

$\mathrm{DT}_{f|_{s_2}}(x_1)$ is the one-dimensional distance transform restricted to row s_2.

Hence, the distance transform of an N-dimensional image can be computed by iterating over the dimensions and performing a one-dimensional transform for each scanline, using the result of the preceeding iteration as function f.

For the next few sections, we restrict our discussion to the one-dimensional case.

2.2 Generalized Squared Euclidean Distance Transform of a Sampled Function

Let $G = \{0, s, \ldots, (n-1)s\}$ be a one-dimensional grid with spacing s and let f be a sampled function $f : G \to \mathbb{R}$.

Then, $\mathrm{DT}_f(x) = \min_{x' \in G}((x-x')^2 + f(x'))$. The graph of the function $p_{x'}(x) = (x-x')^2 + f(x')$ is a parabola with apex at $(x', f(x'))$ and $\mathrm{DT}_f(x)$ finds the lower envelope of all such parabolas (Figs. 1, 2).

This lower envelope is constructed by considering $p_{x'}$ for increasing x':

Say we already have found the parabolas participating in the lower envelope for p_0 up to p_{i-1} (Fig. 2). Let the rightmost parabola in the envelope be p_l.

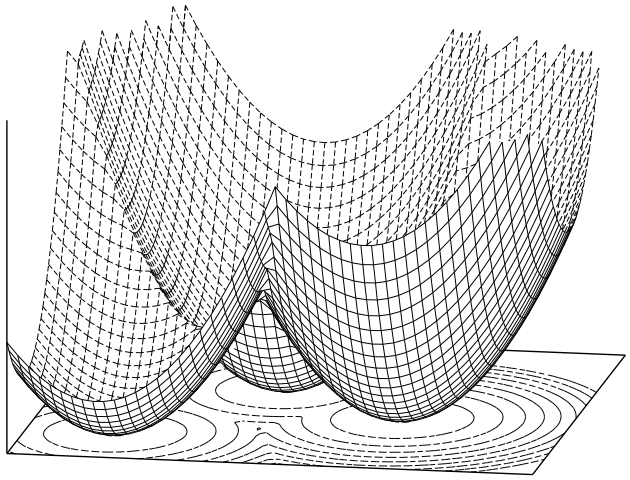


Figure 1: A distance transform with the squared euclidean distance for three points, visualized as a height field. It becomes apparent that this is the minimum of paraboloids.

The next parabola p_i has its apex to the right of any of the other parabolas and will thus be below all of them from some point onwards. We find this point by intersecting p_i with p_l:

$$\begin{aligned}
& p_i(x) = p_l(x) \\
\Leftrightarrow\ & (x-i)^2 + f(i) = (x-l)^2 + f(l) \\
\Leftrightarrow\ & x^2 - 2xi + i^2 + f(i) = x^2 - 2xl + l^2 + f(l) \\
\Leftrightarrow\ & x(2l-2i) = l^2 - i^2 + f(l) - f(i) \\
\Leftrightarrow\ & x = \frac{l^2 - i^2 + f(l) - f(i)}{2(l-i)} \\
\Leftrightarrow\ & x = \frac{1}{2}\left(\frac{l^2 - i^2}{l-i} + \frac{f(l) - f(i)}{l-i}\right) \\
\Leftrightarrow\ & x = \frac{1}{2}\left(l + i + \frac{f(l) - f(i)}{l-i}\right)
\end{aligned} \tag{5}$$

The new parabola p_i will be strictly below p_l in the interval (x, ∞).

We define the *dominance interval* of a parabola as the region where it is strictly below all other parabolas. If x is at or to the left of the dominance interval of p_l, then p_l does not participate in the lower envelope anymore. It has to be removed and the intersection has to be repeated with the parabola preceeding p_l. This process terminates eventually, because the dominance interval of p_0 starts at $-\infty$.

Each parabola will be inserted excactly once and will be removed at most once. Therefore the algorithm has linear complexity.

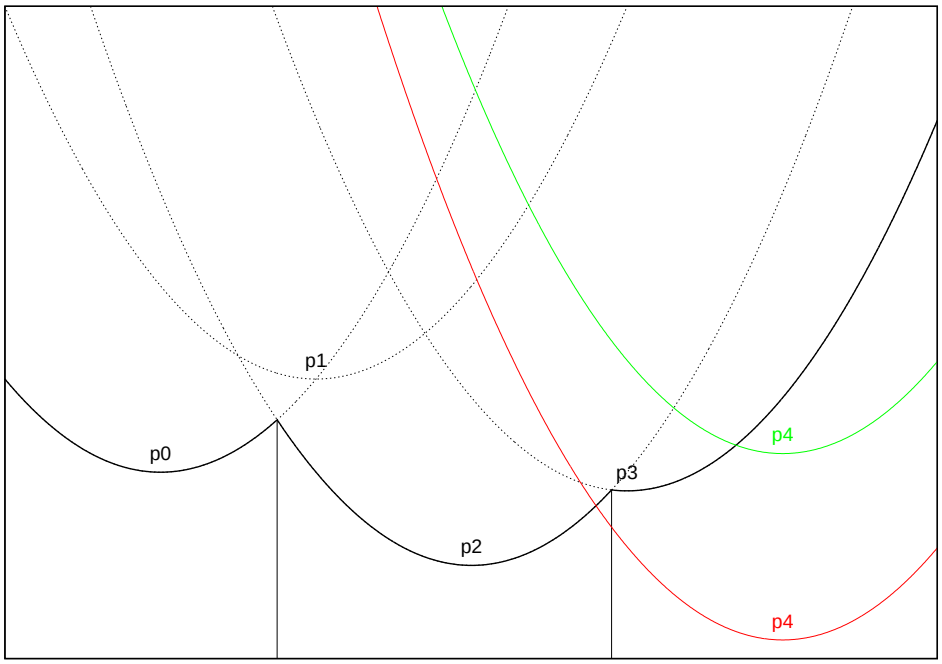


Figure 2: Three of the four black parabolas $p_0, \ldots, p_3$ participate in the lower envelope. If we added the green parabola p_4 with the high apex next, we would not have to delete p_3 from the envelope. If we added the red parabola p_4 with the low apex instead, p_3 would have to be removed from the lower envelope.

When the lower envelope is completed, it represents the *continuous* distance transform of the sampled function f. Uniform sampling of the envelope at points in G has linear complexity as well.

For further details and proofs, we refer the reader to [Meijster et al., 2000, Felzenszwalb and Huttenlocher, 2004].

3 Implementation

We implemented the algorithm as an `itk::ImageToImageFilter` and extended it with the option to create a Voronoi map as well.

To make some optimizations available at compile time, template variables can switch off the use of image spacing and the creation of a Voronoi map. In code, this is implemented by `ifs` whose unused branch is optimized by the compiler during dead-code elimination. This makes it easy to create an alternative implementation which can switch behaviour at runtime.

3.1 itk::LowerEnvelopeOfParabolas

This class template implements the algorithm described in 2.2. Its template parameters are:

`bool UseSpacing` Indicates whether image spacing should be respected or assumed to be 1.

`class TSpacingType` Represents image positions in world coordinates. It must have at least the range of the other numerical types.

`unsigned char MinimalSpacingPrecision` Number of decimal places for the image spacing that can be used with this class.

`bool CreateVoronoiMap` Indicates whether a Voronoi map should be created during computation of the distance transform.

`class TLabelType` Represents voxels in the Voronoi map. If Voronoi maps are used, parabolas are carrying a value of type `TLabelType` with them. During the uniform sampling of the lower envelope, these values are written to the Voronoi map.

`class TAbscissaIndexType` The abscissas of parabola apexes and dominance region borders are stored in this signed type. They are only converted to world coordinates when necessary by multiplication with the spacing. In this way, most of the computation can be carried out in integer arithmetic.

`class TApexHeightType` The ordinates of parabola values are stored in this signed type. In `itk::GeneralizedDistanceTransformImageFilter` this is the type of voxels in the distance image.

Intersection of Parabolas

In section 2.2 we have derived the following formula for the intersection of two parabolas $p_i(x) = (x-i)^2 + f(i)$ and $p_l(x) = (x-l)^2 + f(l)$:

$$x = \frac{1}{2}\left(l + i + \frac{f(l) - f(i)}{l - i}\right) \tag{6}$$

First of all, we carry out all computations on the grid G with uniform spacing s. Let $\tilde{a} := \frac{a}{s}$ for $a \in \mathbb{R}$ in order to let us define parabola apex abscissas in integers:

$$\begin{aligned} & x && = \frac{1}{2}\left(l + i + \frac{f(l) - f(i)}{l - i}\right) \\ \Leftrightarrow \quad & \tilde{x}s && = \frac{1}{2}\left(s(\tilde{l} + \tilde{i}) + \frac{f(l) - f(i)}{s(\tilde{l} - \tilde{i})}\right) \\ \Leftrightarrow \quad & \tilde{x} && = \frac{1}{2}\left(\tilde{l} + \tilde{i} + \frac{f(l) - f(i)}{s^2(\tilde{l} - \tilde{i})}\right) \end{aligned} \tag{7}$$

If $\tilde{l} > \tilde{i}$, then p_l is below p_i in the interval $(\tilde{x}s, \infty)$.

Preventing Arithmetic Overflow

`itk::LowerEnvelopeOfParabolas` defines the static class variables `minimalSpacing`, `maxAbscissa`, and `maxApexHeight` that are used to prevent arithmetic overflows.

`minimalSpacing` is set by the user indirectly via the number of required decimal places. This is specified by the template parameter `MinimalSpacingPrecision`. To prevent floating point underflow and a

division by zero in the intersection computation, we assert that `SpacingType` can represent the square of `minimalSpacing`:

```
assert(0 < minimalSpacing * minimalSpacing)
```

The addition and subtraction of the abscissa indices $\tilde{i}$ and $\tilde{l}$ in equation 7 is carried out in `AbscissaIndexType`. To prevent overflows we set

```
maxAbscissa = numeric_limits<AbscissaIndexType>::max() / 2
```

and assert that all index coordinates are in the range $[-$`maxAbscissa`$,$`maxAbscissa`$]$.

The subtraction $f(l)-f(i)$ in equation 7 is carried out in `ApexHeightType` and therefore we have an analog upper bound for `maxApexHeight`. But since the rest of the computation is carried out in `SpacingType`, there is another bound to be met.

In order to simplify the derivation of `maxApexHeight`, we introduce the following abbreviations:

```
S = numeric_limits<SpacingType>::max()
m = minimalSpacing
A = numeric_limits<AbscissaIndexType>::max()
a = maxAbscissa = A/2
H = numeric_limits<ApexHeightType>::max()
h = maxApexHeight
```

Then we have:

$$\begin{aligned} & \frac{1}{2}\left(\tilde{l}+\tilde{i}+\frac{f(l)-f(i)}{s^2(\tilde{l}-\tilde{i})}\right) \leq a+a+\frac{h-(-h)}{m^2} \leq S \\ \Leftrightarrow \quad & 2a+\frac{2h}{m^2} \leq S \\ \Leftrightarrow \quad & h \leq \frac{m^2(S-2a)}{2} \end{aligned} \tag{8}$$

and can set

```
maxApexHeight = min(H/2, (S-2*a)*m^2/2)
```

These constraints are enforced with `assert()` and have no additional runtime cost in production code.

While arithmetic overflows and underflows can be successfully prevented in the intersection of parabolas, overflows can still happen during the sampling of the lower envelope, causing a wraparound of distance values. If this should be avoided, the user has to provide a `SpacingType` that can hold the largest expected distance value for the images in her application.

Precision

Let us review the intersection formula again:

$$\tilde{x} = \frac{1}{2}\left(\tilde{l} + \tilde{i} + \frac{f(l) - f(i)}{s^2(\tilde{l} - \tilde{i})}\right) \tag{9}$$

The envelope is sampled only at G. Since $(\tilde{x}s, \infty) \cap G = (\lfloor \tilde{x} \rfloor s, \infty) \cap G$, the left interval border index can be represented with the integer $\lfloor \tilde{x} \rfloor$ instead of $\tilde{x}$.

If `AbscissaIndexType`, `ApexHeightType` and `SpacingType` are integer types, then all arithmetic operations will be carried out in integer. Hence, we have

$$\lfloor \tilde{x} \rfloor = \left\lfloor \frac{1}{2}\left(a + \frac{b}{c}\right) \right\rfloor \tag{10}$$

for integers a, b, and c. If we are rounding $\frac{b}{c}$ by truncation, the result can be off by -1. This happens if $\frac{b}{c} < 0$ and $a + \lceil \frac{b}{c} \rceil$ is even.

This error is hardly noticeable in the image, and we did not dedicate special logic to correct it. A floating point type should be chosen as `SpacingType` if full precision is needed.

3.2 itk::GeneralizedDistanceTransformImageFilter

This class template iterates over all dimensions and computes the generalized distance transform for each scanline with `itk::LowerEnvelopeOfParabolas`. The template parameters of `itk::LowerEnvelopeOfParabolas` are exposed to the user. The only mandatory parameters are the types of the input image `TFunctionImage` and output image `TDistanceImage`. All other parameters are either derived from those or have reasonable default values.

The input image is the function whose distance transform will be computed.

If Voronoi maps are generated, which is the default, a second input image must be provided that contains a label for each voxel. See the next sections for examples.

`itk::GeneralizedDistanceTransformImageFilter` is a straightforward implementation of the following pseudocode:

```
for each dimension d:
  for each scanline l:
    l = generalizedDistanceTransform(l)
```

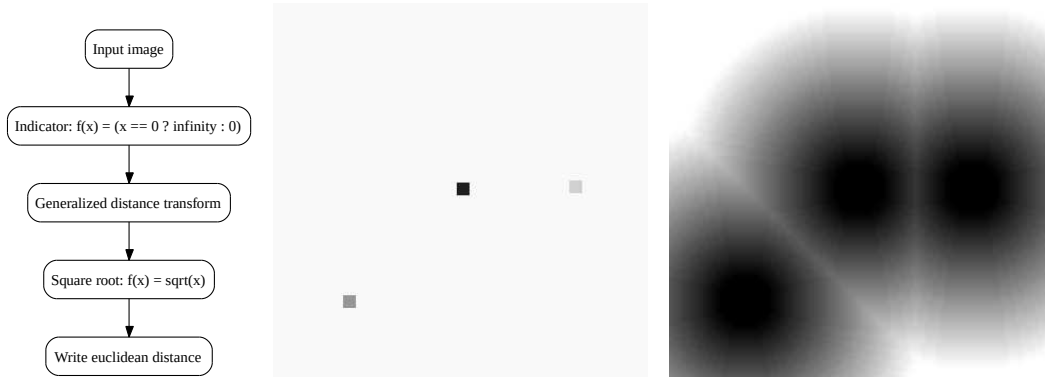
4 Applications

4.1 Euclidean Distance Transform

The algorithm computes the squared euclidean distance transform of a set S if we use the indicator function i_S (see section 2).

In the example program `euclideanDistanceTransform.cxx`, an `itk::BinaryThresholdImageFilter` transforms an input image with background value 0 into such an indicator function.

Figure 3 shows the processing pipeline and the result for a test image.

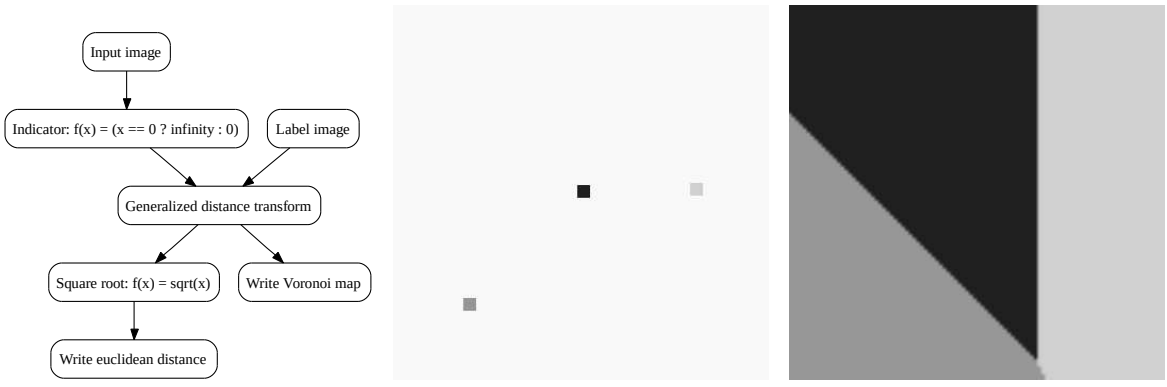


(a) (b) (c)

Figure 3: The pipeline in 3(a) shows how the euclidean distance transform of an image with background value 0 is computed. The input image 3(b) with three foreground voxels is converted to 3(c).

4.2 Voronoi Map

In this example, we add the computation of a Voronoi map. Conceptually, nothing changes except that the second input image is provided and the second result image is written. Figure 4 shows the processing pipeline and the result for a test image.



(a) (b) (c)

Figure 4: The pipeline in 4(a) shows how the Voronoi map of an image with background value 0 is computed. The input image 4(b) has three foreground voxels. The label image assigns a scalar value to each of them, symolized by their grey value. The resulting Voronoi map 4(c) is created simultaneously to the distance transform.

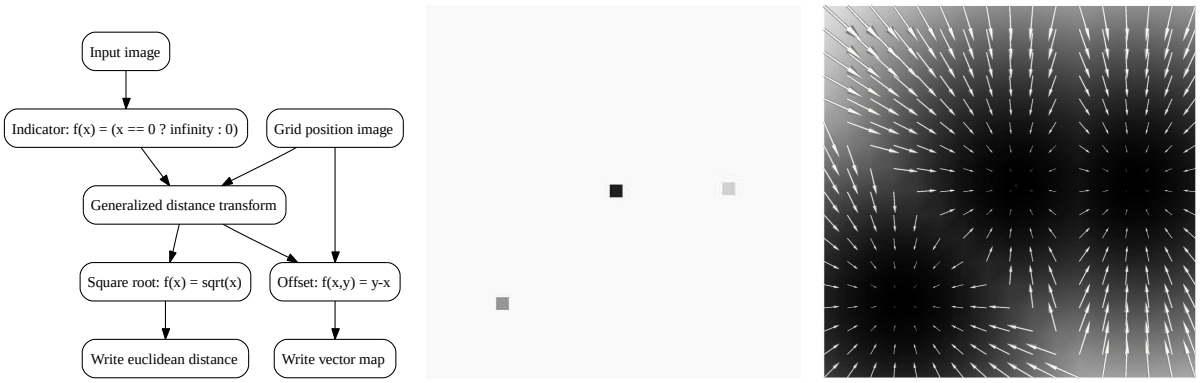
4.3 Vector Map

Sometimes not only the distance to the next foreground voxel is important but also its position. This information is generated by `itk::GeneralizedDistanceTransformImageFilter` if the identity transform of the image domain $D \subset \mathbb{R}^N$ is provided as a label map:

$$g : \begin{cases} D \rightarrow & D \\ x \mapsto & x \end{cases} \tag{11}$$

If V is the resulting Voronoi map, $V - g$ is a map of offsets to the closest voxel.

Figure 5 shows the processing pipeline and the result for a test image.



(a) (b) (c)

Figure 5: The pipeline in 5(a) shows how a map of offset vectors to the closest foreground voxels can be computed with the Voronoi mapping facility. The result for the test image 5(b) is visualized in 5(c), superimposed on the euclidean distance transform. Please note that the length of an arrow in 5(c) is only one tenth of the length of the corresponding offset vector. All offset vectors actually reach to one of the three foreground voxels.

4.4 Signed Euclidean Distance Transform

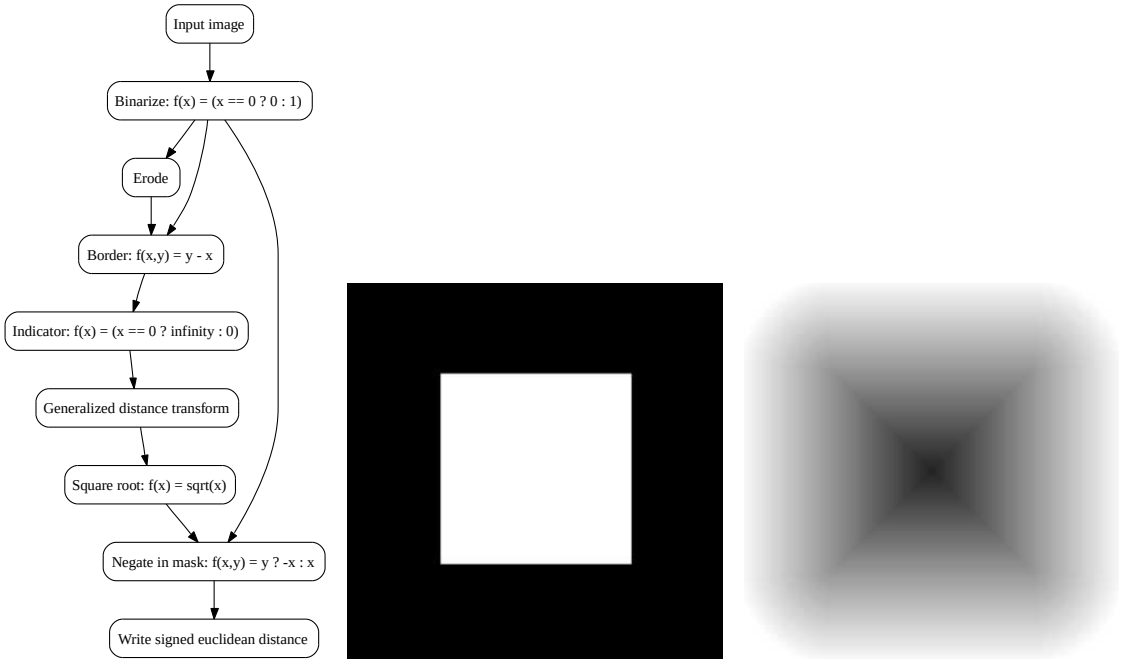
Level set segmenation needs a *signed* euclidean distance transform of a set S, defined by the sum $\mathrm{DT}_S - \mathrm{DT}_{\overline{S}}$. Computation of a distance transform of the border of S and negation of the result inside of S yields the same result. This is the method used in `itk::SignedMaurerDistanceMapImageFilter`, for example.

In the example program `signedEuclideanDistanceTransform.cxx`, the image border is defined by the eroded voxels of a segmentation mask. We provide an `itk::Functor::NegateInMask` in order to perform the negation after the distance transform.

Figure 6 shows the processing pipeline and the result for the test image from Tustison et al. [2006].

4.5 Union of Spheres

Our inital motivation was to create an image filter that computes the union of spheres with different and large radii efficiently. To define the spheres, we wanted to provide an input image r in which the value $r(s) \geq 0$



(a) (b) (c)

Figure 6: The pipeline in 6(a) shows how the signed euclidean distance transform of an image with background value 0 is computed. The input image 6(b) with a rectangular foreground area is converted to 6(c).

of a voxel s is the radius of a sphere centered at s. This is an application of the dilation operator in group morphology with an euclidean sphere structure element and an arbitrary radius function [Cuisenaire, 2006].

The generalized distance transform computes the lower envelope of paraboloids over the image domain D (Fig. 1). The 0-level set of a paraboloid $p_s(x) = (x-s)^2 - r(s)^2$ with apex abscissa s and apex height $-r(s)^2$ is equivalent to the border of the sphere $\{x \mid |x-s| = r\}$.

This leads to the idea of thresholding DT_{-r^2} for values ≤ 0. This does not quite work, however, because the positions s with $r(s) = 0$ would contribute to the lower envelope. To remove them, the apex height has to be set to ∞:

$$f(s) := \begin{cases} \infty & \text{if } r(s) = 0 \\ -r(s)^2 & \text{otherwise} \end{cases} \tag{12}$$

This completes the computation of the union of spheres:

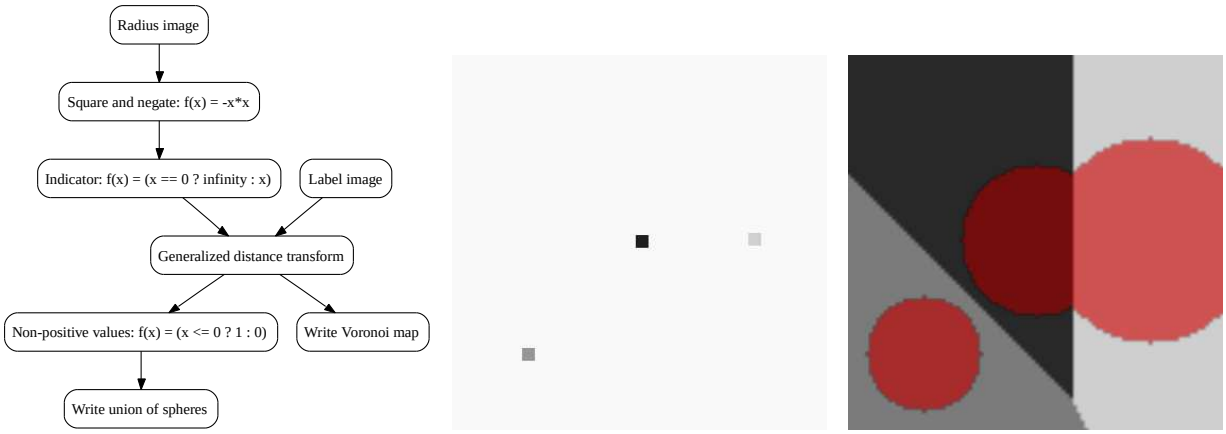
$$\bigcup_{s \in D} \{x \mid |x-s| \leq r(s)\} = \{x \mid \mathrm{DT}_f(x) \leq 0\} \tag{13}$$

Figure 7 shows the pipeline and result for a test image.

We implemented two `itk::Accessor`s that are combined in order to create f. `itk::MinusSqrAccessor` converts a value x to $-x^2$ and `itk::IndicatorAccessor` changes each 0 to a value that represents ∞.

The user of itk::GeneralizedDistanceTransformImageFilter has access to the instance of itk::LowerEnvelopeOfParabolas and especially to the constant maxApexHeight. This value should be used as an indicator for background voxels in equation 12.

Please note that the runtime varies linearly with the number of voxels in the input image and does not depend on the number or radii of the spheres.



(a) (b) (c)

Figure 7: Figure 7(a) shows the pipeline for computing a union of labeled spheres that are defined by a radius image and a label image. The three foreground voxels in the test image 7(b) have the associated radii 15, 20, and 27. The result 7(c) shows the union of the respective spheres superimposed on their Voronoi regions.

5 Performance

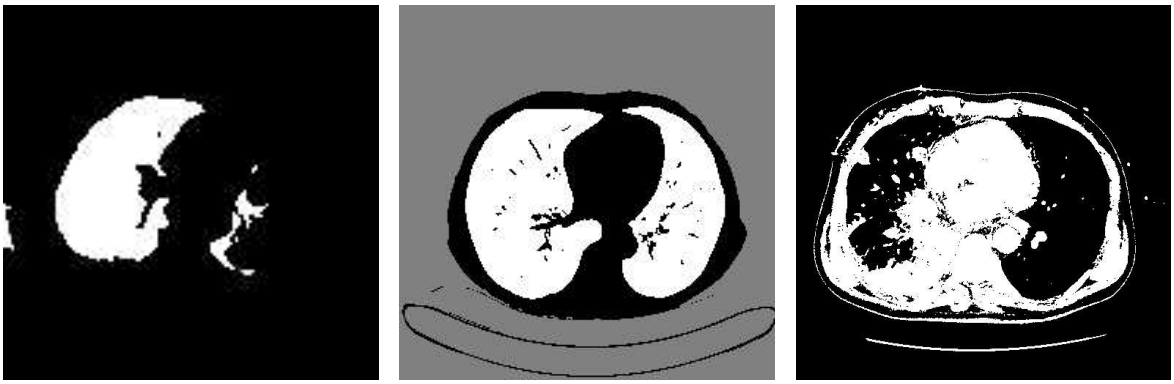
We compare the speed of our implementation with itk::DanielssonDistanceMapImageFilter for three different image sizes. This may be unequable, because itk::DanielssonDistanceMapImageFilter *always* computes Voronoi and vector distance maps and can switch the use of spacing information at runtime in contrast to itk::GeneralizedDistanceTransformImageFilter. We consider this a main feature of our implementation, however, because in our applications we often do not need Voronoi or vector distance maps and were not able to avoid the runtime and memory overhead before.

All performance testing was done on a 2.8 GHz Intel Pentium 4 CPU with a cache of 512 KB running Gentoo Linux. We used gcc-3.4.4 with compile flags `-O3 -fomit-frame-pointer`.

We tested the performance with dataset sizes $136 \times 136 \times 121$, $256 \times 256 \times 165$, and $512 \times 512 \times 348$ (Fig. 8). The datasets are from the pool of cases that we use for the development of clinical applications. The test programs and datasets are distributed with this paper.

5.1 Time

For testing the runtime of our implementation, we loaded a dataset and used it as input for either itk::DanielssonDistanceMapImageFilter or itk::BinaryThresholdImageFilter that acts as an indicator function for itk::GeneralizedDistanceTransformImageFilter.



(a) Small dataset with an intermediate result in lung segmentation.

(b) Medium dataset with structures containing air. Connected components have unique labels.

(c) Full size dataset of a thresholded polytrauma scan.

Figure 8: Datasets for performance tests

After updating the image reader we measured the time it takes to update the distance filter as well. For the `itk::GeneralizedDistanceTransformImageFilter`, this includes the update of the indicator function.

We tested the influence of using spacing information and generating Voronoi maps. Each test has been performed 10 times on an otherwise idle system.

Table 1 details the mean runtime and speedup with respect to `itk::DanielssonDistanceMapImageFilter`. The speedup with Voronoi map generation is between 3.23 and 7.51. If Voronoi maps are disabled, the speedup is between 5.84 and 10.49.

We also measured the time needed by `itk::SignedMaurerDistanceMapImageFilter`, which was recently introduced into ITK by Tustison et al. [2006]. Please note that it computes the *signed* distance transform and should not be compared directly to the other two filters for this reason. See example 4.4 for a method to compute the signed distance transform with `itk::GeneralizedDistanceTransformImageFilter`.

		Image size					
		$136 \times 136 \times 121$		$256 \times 256 \times 165$		$512 \times 512 \times 348$	
	Filter	time (s)	speedup	time (s)	speedup	time (s)	speedup
Spacing enabled	Danielsson	7.37		35.2		306.4	
	Signed Maurer	*4.03*		*16.0*		*177.5*	
	Generalized + V. map	0.98	7.51	7.2	4.87	82.8	3.70
	Generalized	0.81	9.06	5.6	6.28	48.7	6.29
Spacing disabled	Danielsson	6.18		29.6		256.7	
	Signed Maurer	*3.29*		*11.6*		*140.8*	
	Generalized + V. map	0.85	7.25	6.7	4.38	79.4	3.23
	Generalized	0.59	10.49	5.1	5.84	43.4	5.92

Table 1: Runtime and speedup with resepect to `itk::DanielssonDistanceMapImageFilter`. The times for `itk::SignedMaurerDistanceMapImageFilter` are only for orientation and can not be compared directly to the other filters.

5.2 Cache

The algorithm traverses all scanlines in all dimensions. This can lead to cache misses due to non-local memory access.

In order to test the impact of cache misses on our implementation, we have temporarily modified `itk::GeneralizedDistanceTransformImageFilter` to only traverse the scanlines of a single dimension. Table 2 shows the time of traversal for the individual dimensions. For the small dataset, no significant difference in runtime can be observed. For the dataset of medium size, the last iteration takes three times longer than the first. For the big dataset, the second and third iteration both take about three times longer than the first.

We assume that this is due to cache misses. We have used the non-intrusive performance monitoring tool OProfile [Levon et al.] to get an approximization of the L2 and L3 cache misses on our processor. It is apparent from table 3 that an increase in cache misses is correlated with an increase in runtime of our algorithm.

For the big dataset, about 60 % of the runtime can be amounted to inefficient memory access.

Dimension	0	1	2
Small dataset	0.29	0.3	0.34
Medium dataset	1.37	1.45	3.99
Big dataset	11.29	32.76	39.21

Table 2: The time it takes to process all scanlines along a single dimension given in seconds

Dimension	0	1	2
Small dataset	8	84	84
Medium dataset	66	692	42565
Big dataset	392	368078	425517

Table 3: Misses of L2 and L3 cache as reported by OProfile in multiples of 1000

6 Discussion and Future Work

The algorithm to compute the generalized distance transform of a sampled function by Felzenszwalb and Huttenlocher [2004] has been implemented in the ITK framework. Some aspects of the implementation can be tuned at compile time, such as the consideration of image spacing or the creation of Voronoi maps. Our code is rich in comments and adheres to the general style of ITK. Also provided are example programs that demonstrate the use of the filter. We compiled and tested our implementation with Gentoo Linux, Mac OS X 10.3, Windows 2000, and Windows XP. We hope that this minimizes the amount of work for the maintainers of ITK to include our code.

`itk::GeneralizedDistanceTransformImageFilter` computes a larger class of distance transforms than `itk::DanielssonDistanceMapImageFilter` and achieves a speedup of up to 10.49 in our tests. Although not reported in this paper, we could achieve even better speedups for a 1.5 GHz PowerPC G4 processor with 512 KB cache. We are comfortable with this performance, but we also believe that more significant improvements would be possible if the amount of non-local memory access could be reduced.

Our current and future lines of work are twofold:

- We implemented two additional algorithms to compute the union of spheres, that both rest upon the duality between a sphere and the 0-level set of a spherical paraboloid. The first one computes the union of K spheres at arbitrary locations with a complexity of $O(KM^{(N-1)})$ for an M^N image. The second one further expands this idea by also maximizing a constant value that is associated with the spheres.

  These algorithms have also been implemented within the context of ITK but not as `itk::ImageToImageFilter`s yet.

- Motivated by our observations regarding the memory access behaviour of our implementation, we are currently considering different memory layouts for images than ITK's default row-major order. This might also prove beneficial for other algorithms with memory access patterns similar to ours, hence we plan to implement an `itk::ImageToImageFilter` that can reorder images accordingly.

References

Olivier Cuisenaire. *Distance Transformations: Fast Algorithms and Applications to Medical Image Processing*. PhD thesis, Laboratoire de Telecommunications et Teledetection, Université Catholique de Louvain, 1999. 1

Olivier Cuisenaire. Locally adaptable mathematical morphology using distance transformations. *Pattern Recognition*, 39(3):405–416, March 2006. URL http://lts1pc19.epfl.ch/repository/Cuisenaire2005_1276.pdf. 1, 4.5

P. E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980. 1

Pedro F Felzenszwalb and Daniel P Huttenlocher. Distance transforms of sampled functions. Technical report, Cornell University, September 2004. 1, 2, 2.1, 2.2, 6

Y. Ge and J.M. Fitzpatrick. On the generation of skeletons from discrete euclidean distance maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:1055 – 1066, 1996. 1

John Levon et al. OProfile: A system-wide profiler for linux systems. URL http://oprofile.sourceforge.net. 5.2

Calvin R. Maurer, Rensheng Qi, and Vijay Raghavan. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):265 – 270, 2003. 1

Arnold Meijster, Jos B.T.M. Roerdink, and Wim H. Hesselink. A general algorithm for computing distance transforms in linear time. *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 331–340, 2000. 2.2

J. B. T. M. Roerdink and Arnold Meijster. The watershed transform: Definitions, algorithms and parallelization strategies. *Fundamenta Informaticae*, 41(1 – 2):187 – 228, 2000. 1

J.A. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, 1996. 1

Jun-ichiro Toriwaki and Kensaku Mori. Distance transformation and skeletonization of 3d pictures and their applications to medical images. *Digital and Image Geometry: Advanced Lectures*, pages 412 – 428, 2001. 1

Nicholas J. Tustison, Marcelo Siqueira, and James C. Gee. N-D linear time exact signed euclidean distance transform. *The Insight Journal*, Feb 2006. 1, 4.4, 5.1