Utrecht University

Department of Information and Computing Sciences

Generalized Distance Transforms using the FEED-class Algorithm

BSc
Computer Science

Author:
Gerben Aalvanger (3987051)

Supervisor:
dr dr. Egon L. van den Broek

August 28, 2015

Generalized Distance Transforms using the FEED-class Algorithm

G.H. Aalvanger

August 28, 2015

Abstract

The Fast Exact Euclidean Distance (FEED) transform is generalized to support intensity values and gray-scale images. The generalized FEED (gFEED) class algorithms support both Euclidean and squared Euclidean distances. The algorithms are tested on datasets by Schouten and van den Broek and on newly developed datasets. Tests show a $\mathcal{O}(m \times n)$ time complexity on squared Euclidean distances and a $\mathcal{O}(m \times n)(m + n)$ time complexity on Euclidean distances.

Keywords. (Generalized) Distance transforms, (Squared) Euclidean distance, Sampled function, Fast Exact Euclidean Distance (FEED), linear, generalized FEED (gFEED)

1 Introduction

A Distance Transforms (DT) transforms an image or grid to a new image or grid, called a Distance Map (DM). It is used in image processing and robotics [13]. It computes the distance of a (specified) set of elements in a grid, to the closest element from another specified set of elements in the grid. For binary images, object and background pixels are defined as elements in the grid. The DT on background pixel p in the image is the distance d to the closest object pixel $q \in O$:

$$D(p) = \min_{q \in O} (d(p, q)), \tag{1}$$

with distance function $d(p, q)$ often defined as the Euclidean Distance (ED) being

$$d_{Eucl}(x, y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}, \tag{2}$$

on n-dimensional tuples x and y. However, other distance functions can be adopted, like the Manhattan distance (also known as city-block or L_1 distance) and the squared Euclidean Distance (ED^2), which does not satisfy triangle inequality. This paper discusses the ED and the ED^2. Further, we limit our work to squared grids; triangular and hexagonal grids are not discussed.

DT algorithms can be classified in four broad classes [9], based on[4]:

1. Ordered propagation: These algorithms start from object pixels and progressively determine new distance values to background pixels in order of increasing distance.[2]

2. Raster scanning: The image is scanned line by line, top to bottom and, subsequently, in reversed order using 2D masks.[1][11][12]

3. Independent scanning: Each row and column of the image is processed independently of each other. Either the rows or the columns can be scanned first. After the horizontal and vertical scan, the final DT is produced. The values of the intermediate image cannot be regarded as distances.[5][8]

4. FEED Algorithms: Eq. 1 suggests that background pixels 'eat' their minimum distance to an object pixel. Instead of this, the object pixels 'feed' their distance to the background pixels. The number of background pixels an object pixel should feed is reduced by various optimization strategies. [9][10]

Traditionaly, these classes of DTs are limited to binary images; however, DTs can be generalized to sampled functions [5][7]:

$$D_f(p) = \min_{q \in G} (d(p, q) + f(q)), \tag{3}$$

where $f(q)$ is a function over element q, which returns a rational number from $\mathbb{Q}$. This generalized DTs (gDT) supports gray-scale images and multi-valued grids. Without this generalization, only the absence or presence of a feature at each pixel can be specified. The generalization can specify a cost for a feature at each pixel. Now, the distance transform reflects a combination of distance and feature costs. Using this approach, one can for instance split an image in two regions, using two groups of pixels having different 'strengths'. Each region is influenced by one of the two pixel groups. The gDT can also be applied as a geodesic closing or opening operator [3], this is useful for image segmentation and edge-sensitive smoothing. The gDT has also been used for inference using large state-space hidden Markov Models, because of the similarity between transition probabilities and distance functions [6].

In this paper, we introduce the generalized FEED (gFEED) algorithm. We start with a brief introduction of the Fast Exact Euclidean Distance (FEED) algorithm and its relevant optimizations (Section 2). In Section 3 and 4, gFEED is introduced for the ED^2 and ED. Section 5 approaches the performance of the algorithms in terms of exactness, correctness and time complexity. In Section 6, gFEED is tested and compared to state of the art algorithms on multiple grids with different properties. In Section 7, the results of this paper are discussed and further research is proposed.

2 Principles of FEED

The FEED-algorithm is derived from the inverse definition of the DT (Eq. 1). Instead of looping through the background pixels to find the minimum distance to an object pixel, FEED loops through the object pixels and they 'feed' their distance to all background pixels.

Data: O: object pixels, B: background pixels
Result: Distance Map (DM), with distance transform on all background pixels
Initialize DM;
foreach $q \in B$ **do**
| $DM(q) \leftarrow \infty$
end
Start DT;
foreach $q \in O$ **do**
| **foreach** $p \in B$ **do**
| | $DM(p) \leftarrow min(DM(p), d_{Eucl}(q,p))$
| **end**
end
.

Algorithm 1: Basic FEED

FEED knows many optimization strategies, needed for runtime reduction. Here, we discuss the three strategies that are used for gFEED.

The first and easiest strategy is limiting the amount of 'feeding' object pixels. The distance to a pixel within an object is longer than the distance to the closest borderpixel of that object. Therefore, only border objectpixels $B(O)$ are feeding their distance to the background pixels.

The second strategy limits the amount of background pixels that an object pixel should feed. Given a feeding border pixel b and a random object pixel q, a line between b and q can be drawn. Perpendicular to this line, a bisection line can be drawn between the two pixels. The pixels on one side of the line are closer to b and pixels on the other side are closer to q. Therefore, the pixels to be fed by b are limited to those that are on his side of the bisection line. By drawing more bisection lines, a complex shape is constructed, which contains the pixels to be fed by b. For an efficient representation of this shape, the bisection lines are defined in a new coordinate system $\bar{c}$: $(x,y) \rightarrow (x - b_x, y - b_y)$. Now the feeding pixel b is shifted to the origin: $\bar{b} = (0,0)$ and $\bar{q} = (q_x - b_x, q_y - b_y)$. The bisection line between $\bar{b}$ and $\bar{q}$ is defined by all points $\bar{p} = (x,y)$ in $\bar{c}$ having

$$\begin{aligned} d_{Eucl}(\bar{b},\bar{p}) &= d_{Eucl}(\bar{q},\bar{p}) \\ \sqrt{(\bar{b_x} - x)^2 + (\bar{b_y} - y)^2} &= \sqrt{(\bar{q_x} - x)^2 + (\bar{q_y} - y)^2}. \end{aligned} \tag{4}$$

Since $\bar{b}$ is shifted to the origin, only the position of $\bar{q}$ should be taken into account:

$$\begin{aligned} \sqrt{x^2 + y^2} &= \sqrt{(\bar{q_x} - x)^2 + (\bar{q_y} - y)^2} \\ x^2 + y^2 &= \bar{q_x}^2 + x^2 - 2\bar{q_x}x - 2\bar{q_y}y + y^2 + \bar{q_y}^2 \\ 2\bar{q_x}x + 2\bar{q_y}y &= \bar{q_x}^2 + \bar{q_y}^2. \end{aligned} \tag{5}$$

If pixel q is selected at random, it is likely not to be the best choice. Every pixel on the line between b and q is a better choice, since it moves the bisection line closer to b. Therefore, a better approach for the selection of q, is to create a scan-line (m,n) with $greatestCommonDivisor(m,n) = 1$ starting in b. Here, (m,n) can be either random or pre-selected. A bisection line is created with the first object pixel the line hits. This is done by calculating the smallest integer

k, where $k(m,n)$ is an object pixel. When $k(m,n)$ is outside the image, no bisection line can be created using (m,n). This process is called line-scanning.

The third strategy simplifies the use of bisection lines. Instead of creating a complex shape, a bounding box is created around the complex shape. The rectangular representation can be efficiently reduced by the bisection lines. However, the bounding box is often larger than the complex shape. Finally, for each row in the resulting bounding box, the intersections with bisection lines can be used to determine the pixels to be fed by $\bar{b}$. This process is not used on small bounding boxes; feeding all pixels in the bounding box is computationally more efficient in this case.

3 Squared Euclidean distance transforms

A gDT differs from how a DT is used. Where a DT only has two non-overlapping classes with 2 intensities (0 and ∞), a gDT could have all values in between them. Therefore, not every strategy used in FEED is always relevant for gFEED algorithms.

Border pixel selection is only useful when clustered pixels with the same intensity occur in the image. An object with varying intensities can feed with low-intensity elements in its core as well as with high-intensity elements on its border. Since background pixels (intensity = ∞) are not necessarily present in the image, gFEED algorithms need to be able to calculate the gDT for non-background pixels as well. In this case, the initial value of a pixel is equal to the intensity of that pixel.

The most important speed-up of FEED using bisection lines can also be used in the computation of the gDT using ED. The bisection line in gFEED using ED^2 ($gFEED^2$) is defined in $\bar{c}$ as all points $\bar{p} = (x,y)$, where

$$\begin{aligned} d_{Eucl^2}(\bar{b},\bar{p}) + f(\bar{b}) &= d_{Eucl^2}(\bar{q},\bar{p}) + f(\bar{q}) \\ (x^2+y^2) + f(\bar{b}) &= ((\bar{q_x}-x)^2 + (\bar{q_y}-y)^2) + f(\bar{q}), \end{aligned} \tag{6}$$

Rewriting this equation gives a similar representation as Eq. 5, for a straight bisection line:

$$2\bar{q_x}x + 2\bar{q_y}y = \bar{q_x}^2 + \bar{q_y}^2 - f(\bar{b}) + f(\bar{q}), \tag{7}$$

which is similar to the line FEED uses as bisection line. Note that for $f(\bar{b}) = f(\bar{q})$ this is equal to normal FEED. Line scanning in the same way as FEED might not be useful, for the same reason border pixel selection is not always useful. Therefore, two new approaches are suggested besides normal line scanning and random selection:

Exhaustive line scan Instead of scanning until the first object pixel, all pixels until the image border are scanned. The best bisection line created is used to reduce the bounding box. Since $\bar{q} = k(m,n)$, the bisection line can be written as

$$2mx + 2ny = \frac{(km)^2 + (kn)^2 - f(\bar{b}) + f(\bar{q})}{k}. \tag{8}$$

The best bisection line is bisection line having the lowest right-hand-side of this equation.

Limited line scan This line scan is the similar to the exhausting line scan; but, instead of stopping at the border, the amount of scanning steps is limited. The scan limit can be set by hand, or the limit can be derived from the currently best known bisection line and the minimum intensity used in the image.

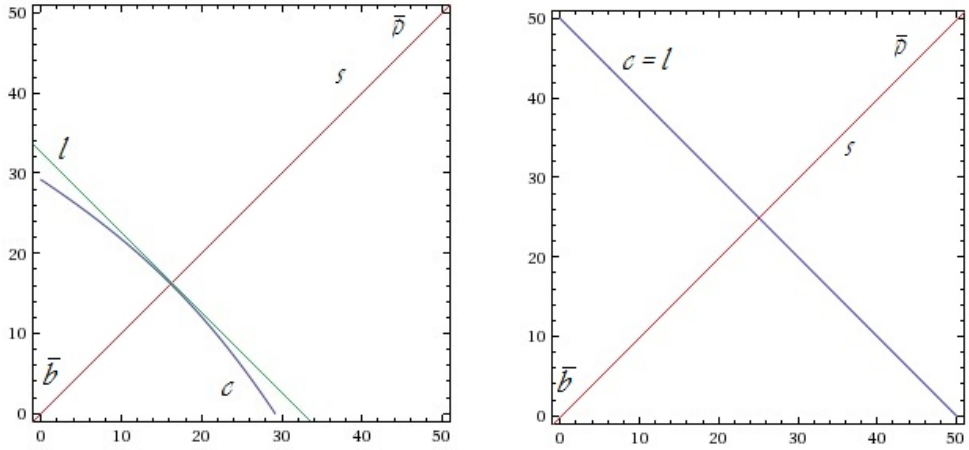
Using gFEED^2 triangle inequality does not hold consequently, unexpected results may appear. For instance, Eq. 7 states that for two points a bisection line can be drawn. If the ED^2 between b and q is smaller than the difference between $f(b)$ and $f(q)$, the bisection line is drawn behind b or q.

4 Euclidean distance transforms

Although computationally more expensive, using ED over ED^2 has advantages, since triangle inequality does not hold for ED^2. Therefore, this section will discus gFEED using ED (gFEED^1) instead of ED^2. As with gFEED^2, more than two values can occur in gFEED^1 and consequently, the first object pixel found on a line scan still does not necessarily give the best bisection line. However, bisection lines in gFEED^1 differ from the bi-section lines used in gFEED^2.

In $\bar{c}$, a point $\bar{p} = (x, y)$, is on the bisection between $\bar{b}$ and $\bar{q}$ if

$$\begin{aligned} d_{Eucl}(\bar{p}, \bar{b}) + f(\bar{b}) &= d_{Eucl}(\bar{p}, \bar{q}) + f(\bar{q}) \\ \sqrt{x^2 + y^2} + f(\bar{b}) &= \sqrt{(\bar{q_x} - x)^2 + (\bar{q_y} - y)^2} + f(\bar{q}). \end{aligned} \tag{9}$$



(a) $f(\bar{q}) < f(\bar{b})$ (b) $f(\bar{q}) = f(\bar{b})$

Figure 1: Bisection between points $b = (0, 0)$ and $q = (50, 50)$ using the euclidean distance.

This equation is not linear and therefore not a bisection line, but a hyperbolic curve C, as can be seen in Figure 1a. If $f(\bar{b}) < f(\bar{q})$ the curve is curving towards $\bar{b}$; so, the area under the curve is strictly smaller than the area limited by the bisection line L. This line is perpendicular to the scanline S and intersects S and C on their intersection. Using this bisection line instead of the curve is

possible as long as $f(\bar{b}) \leq f(\bar{q})$. Since for $f(\bar{q}) = f(\bar{b})$ the hyperbola becomes linear (Figure 1b) and for $f(\bar{q}) > f(\bar{b})$ the curve is moving towards $\bar{q}$.

When calculating the intersection of C and S, the hyperbola causes rounding problems. Square root evaluation results in non-exact numbers and one cannot easily guarantee the correctness of the algorithm. Therefore L will be approximated. For this approximation, an image of $\bar{q}$ is used: $\bar{q'} = k'(m, n)$ and $f(\bar{q'}) = f(\bar{b})$. This can be achieved by moving $\bar{q}$ on the scanline over a distance of $f(\bar{b}) - f(\bar{q})$. The bisection line L' between $\bar{q'}$ and $\bar{b}$ is equal to L. This approach however can result in k' becoming non-integer. To avoid this, k' is rounded up, which implies that L' also moves away from $\bar{b}$.

This approach ensures that rounding does not imply incorrectness, since the area limited by L' is strictly larger than the area limited by C. It also ensures that point $\bar{q'}$ is created with $f(\bar{q'}) = f(\bar{b})$. This results in a bisection line of:

$$2\bar{q_x}x + 2\bar{q_y}y = \bar{q_x}^2 + \bar{q_y}^2 - f(\bar{b}) + f(\bar{q'}) \tag{10}$$

Given that $f(\bar{b}) = f(\bar{q'})$, Eq. 5 can be used. Like gFEED^2, both exhaustive and limited line scans can be used.

When using the bisection strategy, one might notice that k' can become less than 0 when $f(\bar{b}) - f(\bar{q}) > D_{Eucl}(\bar{b}, \bar{q})$. Instead of drawing the bisection line behind $\bar{b}$ as done in gFEED^2, no bisection line is drawn at all. Because of triangle inequality, every pixel in the image will rather be fed by $\bar{q}$ than by $\bar{b}$. We say that $\bar{b}$ is dominated by $\bar{q}$ if:

$$f(\bar{b}) - f(\bar{q}) \geq D_{eucl}(\bar{b}, \bar{q}) \tag{11}$$

When scanning along the scan line, a test should be performed whether Eq. 11 holds. If so, $\bar{b}$ should not feed at all.

5 Results

5.1 Exactness and correctness

In Section 3 and 4, gFEED^2 and gFEED^1 are introduced. Since gFEED^2 starts with rational numbers and calculations only use division, multiplication, addition and subtraction, gFEED^2 does not suffer from rounding problems. Section 3 shows that limiting the bounding box according to Eq. 6, will not result in 'forgotten' pixels. Therefore, gFEED^2 does provide an exact and correct for the gDT using ED. When using the heuristic limit, a square root is evaluated. This does not negatively affect the correctness or exactness of the program, since it can only result in drawing more or less bisection lines.

gFEED^1 also uses rational number calculation, but it approximates bisection lines by square root evaluation. This approximation however is rounded towards the bigger value of k, resulting in correct bisection lines. However, when feeding a pixel, a distance of $f(b)+d(b, q)$ is fed. Therefore, the correctness and exactness of gFEED^1 in pixel q depends on the possibility of determining whether $f(b) + d_{Eucl}(b, q) \leq f(b') + d_{Eucl}(b', q)$. When approximating the square root in the ED with an error of ϵ, the DM is also approximated with an error of ϵ. The same problem of exactness occurs in the basic $O(N^2)$ algorithm.

5.2 Theoretical time complexity

gFEED^2 using the exhaustive line scan method has a time complexity of $\Omega((m \times n) \times (m+n))$ on images of size $m \times n$, since for each pixel $(m \times n)$ an exhaustive scan in x- and y-direction is performed. The time complexity of the (heuristic) limited line is harder to prove and will be discussed in Section 5.3, from an experimental point of view.

gFEED^1 using the exhaustive line scan method has a time complexity of $\Omega((m \times n) \times (m + n))$, because it also uses the exhaustive scan in x- and y-direction. Heuristic and limited line scans have a worst case time complexity bound from below by $\Omega((m \times n) \times (m + n))$. This worst case scenario is based on an image having

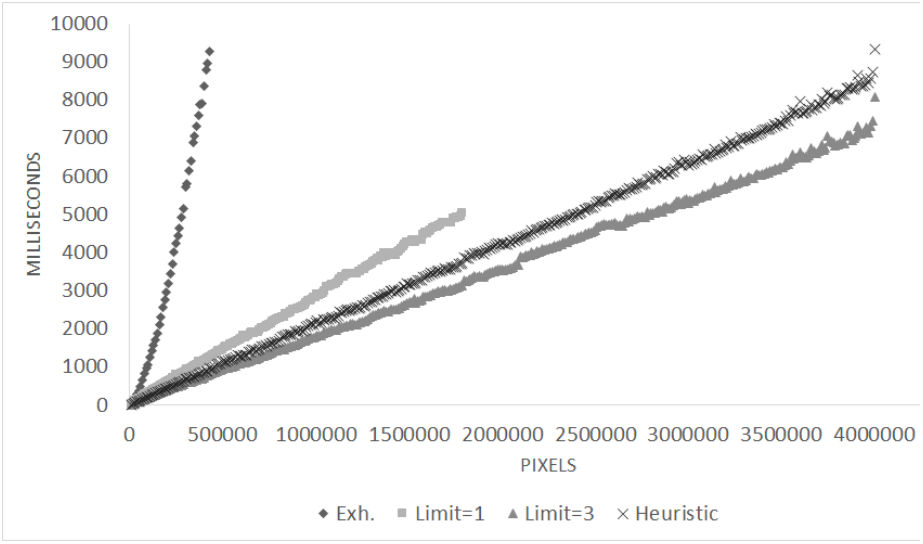
1. values of pixels in rows and columns in decreasing order;
2. a difference in value between adjacent pixels less than 1.

Given this $n \times m$ image, for pixel $p = (x, y)$, the image is scanned in x-direction for a distance of $x - 1$ and in y-direction for a distance of $y - 1$, since no pixel q will be found having $f(q) < f(p)$. In opposite x- and y-direction, no domination occurs, which is ensured by property 1 of the image. For all pixels the amount of scanned pixels is given by $\sum_{i=0}^{n} \sum_{j=0}^{m} (i + j) = \frac{1}{2}(1 + m)(1 + n)(m + n)$, which results in the time complexity of $\Omega((n \times m) \times (n+m))$. The average time complexity for gFEED on ED will be experimentally tested in Section 5.3.

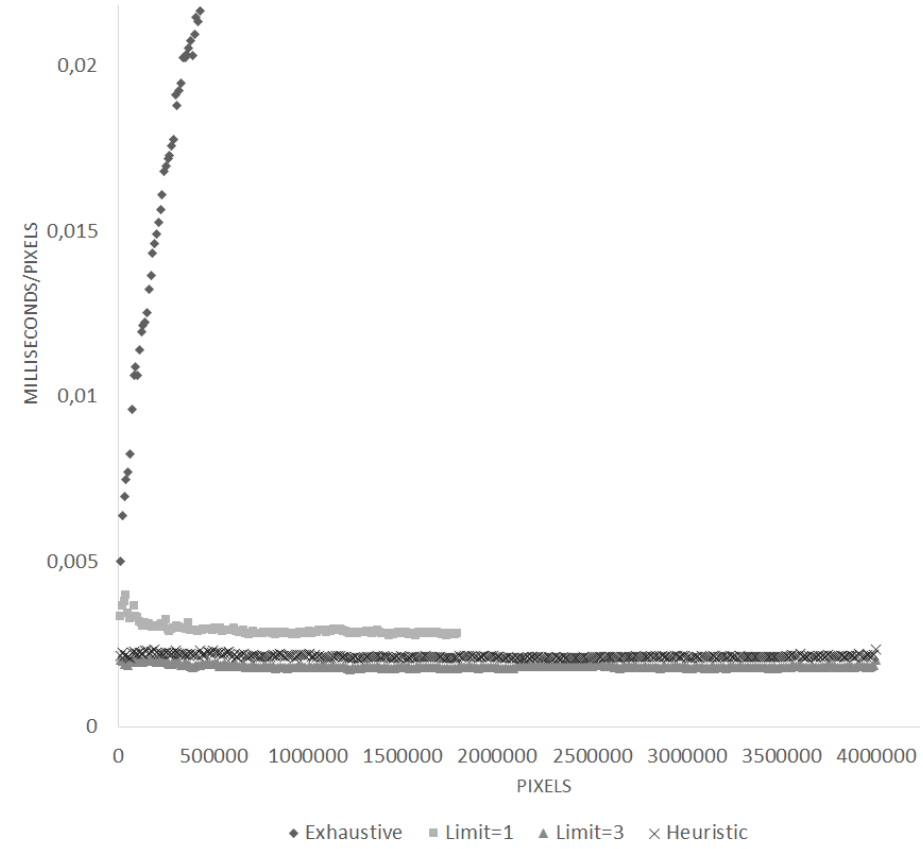
5.3 Experimental time complexity

The time complexity of FEED has been shown to be linear in the amount of image pixels [9], though no formal proof has been given. gFEED^2 has been proven to have a time complexity of $\Omega((m \times n) \times (m + n))$, when using the exhaustive line scan. For the complexity of the limited line scan and the heuristic line scan, an experimental approach is used. gFEED^2 using the exhaustive, limited and heuristic line scan has been tested on a dataset of 4000 images. The square images have 400 different sizes between 100×100 and 400×400. For each size, 5 images with random dots are 5 images containing gray-scale objects (rectangles and ellipses) are created. Figure 2 shows the testing results. The time of the exhaustive line scan grows fast on bigger images, in contrast to the limited and heuristic line scans. Figure 2a suggest a linear time complexity for the limited and heuristic lines scans. Figure 2b confirms this and shows that the exhaustive line scan grows faster than linear. The 1-limited, 3-limited and heuristic line scan have a time/pixel ratio of respectively 0.0029, 0.0018 and 0.0021 milliseconds per pixel. Standard deviations of these three line scans are below 10^{-3}.

gFEED^1 using the exhaustive, limited and heuristic line scan has also been tested on the images in the 4000-image dataset. Because gFEED^1 is slower than gFEED^2, fewer images from the dataset have been used. The results of this tests are shown in Figure 3. The curve in Figure 3a shows a more than linear time growth. Section 5.2 shows how the worst case scenario of is bounded below by $\mathcal{O}(m \times n)(m + n)$. When scaling Figure 3a by $(m \times n)(m + n)$, a constant factor appears, as can be seen in Figure 3b. Therefore, we can conclude that the $\mathcal{O}(m \times n)(m + n)$ complexity also occurs in the average case.

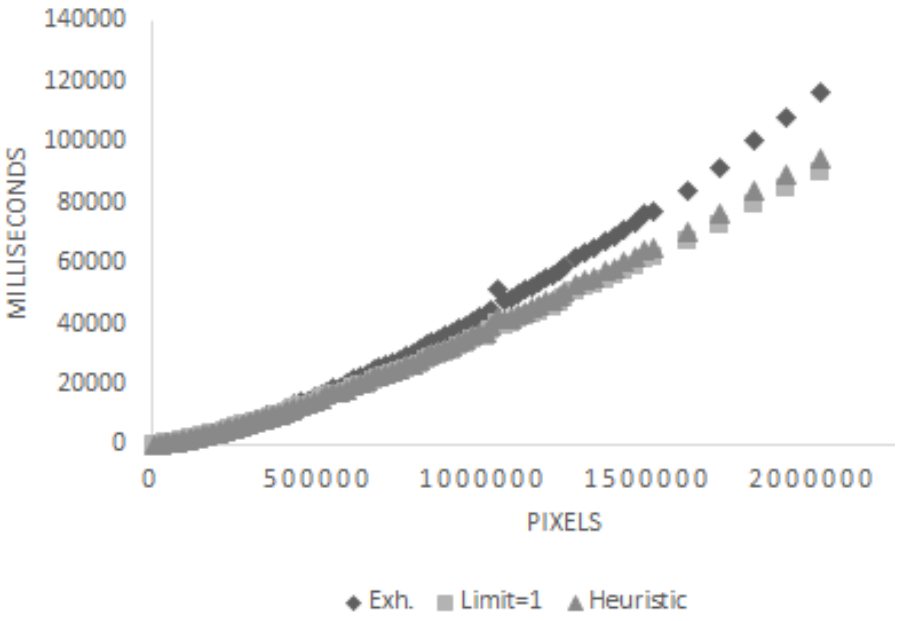


(a) Time-pixel diagram

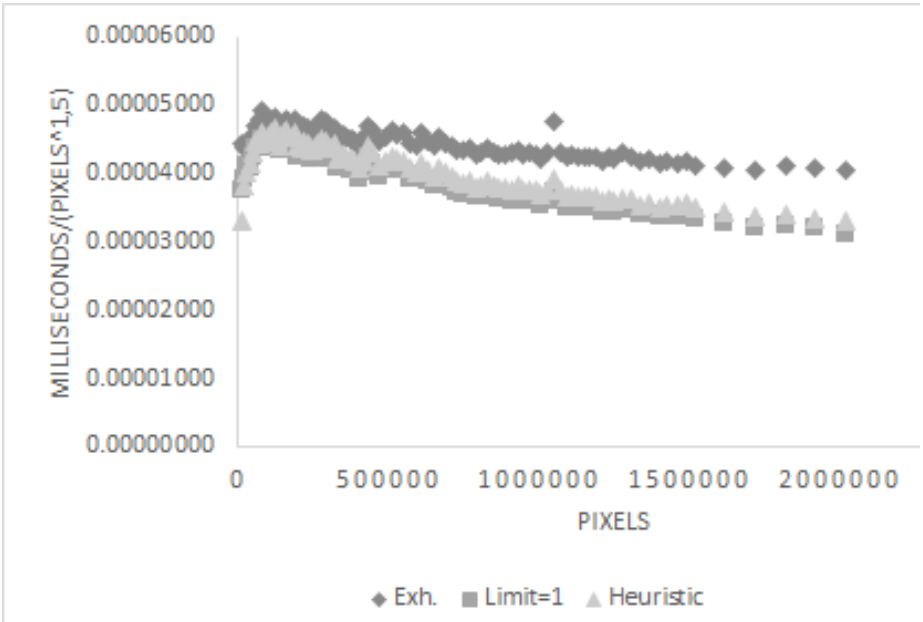


(b) Linear times for limited and heuristic line scans.

Figure 2: Timing results of gFEED[2]



(a) Time-pixel diagram



(b) $\mathcal{O}(m \times n)(m+n)$ complexity for limited and heuristic line scans.

Figure 3: Timing results of gFEED[1]

6 Benchmark

In this section, the different line scans of gFEED[1] and gFEED[2] are experimentally compared. If possible, state of the art algorithms are included for reference. For the comparison, two datasets are used.

The first dataset is generated by Schouten en Van den Broek, and consists of four sub-sets of 32 black and white object-like images, having increasing densities, the size of all images is 1000x1000.

The second dataset is a new generated dataset consisting of 8 subsets with different properties. The first subset consists of 21x20 random images generated with densities between 0 and 100 (stepsize = 5). Each colored pixel has a random gray color. The second, third and fourth subset consist of random (open and/or filled) rectangles and ellipses having random rotations. The last four subsets fill rectangles and ellipses with a linear or radial gradient. Two subsets have noise in the image. All images have a resolution of 500x500 and the datasets consist of 100 images with increasing amount of objects or density.

gFEED algorithms are executed with line scans in directions (m, n) with $|m| \in \{1, 3, 4\}, |n| \in \{1, 3, 4\}$ and $greatestCommonDivisor(m, n) = 1$. Exhaustive, limited and heuristic line scans are used in comparing gFEED versions.

6.1 Squared Euclidean Distance

Table 1: Timing results of the GDT on squared Euclidean Distances

Dataset	Specification	Exh.	Limit=1	Heur.	LLT	PE
Random dots	500x500	5880,3	3679,8	612,8	83,1	86,1
Filled	500x500	2017,5	697,5	392,5	64,2	65,1
Open	500x500	1279,9	562,1	333,9	68,1	68,4
Filled/Open	500x500	1899,9	721,4	427,6	73,4	69,3
Lin. Gradient	500x500	2166,3	744,2	426,1	67,9	71,6
Rad. Gradient	500x500	2258,9	753,9	427,0	71,5	68,4
Lin. Grad. Noise	500x500	4477,9	1128,2	543,5	54,8	51,9
Rad. Grad. Noise	500x500	3845,4	1037,4	494,3	61,1	53,2
Objects	1000x1000	29681,0	7260,0	1862,0	339,5	343,1
Objects (r)oughened	1000x1000	29988,2	7386,8	1850,3	346,1	357,6
Objects (o)verlap-removal	1000x1000	22539,3	5620,8	1788,3	340,5	350,4
Objects o-r	1000x1000	20837,7	5283,3	1624,0	321,2	334,1

Average timing results (in ms) of three gFEED[2] algorithms and two state of the art algorithms.

For time comparison on gFEED[2], two state of the art algorithms as proposed by Lucet [7] and by Felzenszwalb and Huttenlocher[5] are included in the timing of the algorithms. Lucet's algorithm is based on the computation of the Linear-time Legendre Transform (LLT) and Felzenszwalb's algorithm is based on the computation of the Parabolic Envelope (PE). Both algorithms are independent scanning algorithms and compute the exact DM in linear time.

The timing results of gFEED[2] with exhaustive, limited and heuristic line scans, compared to the timing results of both LLT and PE are shown in Table

1. gFEED^2 using the heuristic line scan is always by far the fastest gFEED^2 algorithm. The heuristic scan minimizes the bounding box while also minimizing the line scan, therefore it successfully combines the advantages of the limited and exhaustive line scan. Although the heuristic line scan calculates the DM in less than a second, the LLT and PE algorithm are a factor 6 faster than gFEED^2, this factor is slightly lower in the 1000x1000 images.

The results show that in contrast to FEED, gFEED^2 is slower than LLT and PE. This difference can be attributed to the difference between FEED vs. gFEED^2 and LLT/PE on binary images vs. LLT/PE on gray-scale images. LLT and PE on binary images are not optimized for binary images, but are equal to LLT and PE on gray-scale images (except the final square root is not taken). Therefore, the speed of those algorithms is not negatively influenced when using them on the gDT. gFEED^2 algorithms do differ from the FEED algorithms. FEED algorithms are optimized for binary images: non-border pixels are ignored and line scans are limited to the first object pixel it hits. The gFEED^2 algorithms drop some of this optimizations (border pixel selection) or change the way it is computed (heuristic or limited line scans). Since less optimizations are used in gFEED^1 and others are computed in different (slower) ways, gFEED^1 is slower than FEED, LLT and PE.

Another interesting observation is the relatively slow performance of gFEED^2 on the random dots images. The same is observed when noise is added on the gradient images. The lack of bigger objects with one intensity, also results in less elimination of their non-border pixels.

The three line scan options are also compared according to their behavior on different object pixel densities. For testing, the 'Random dots' dataset is used and results are plotted in figure 4. The heuristic line scan is not affected by the

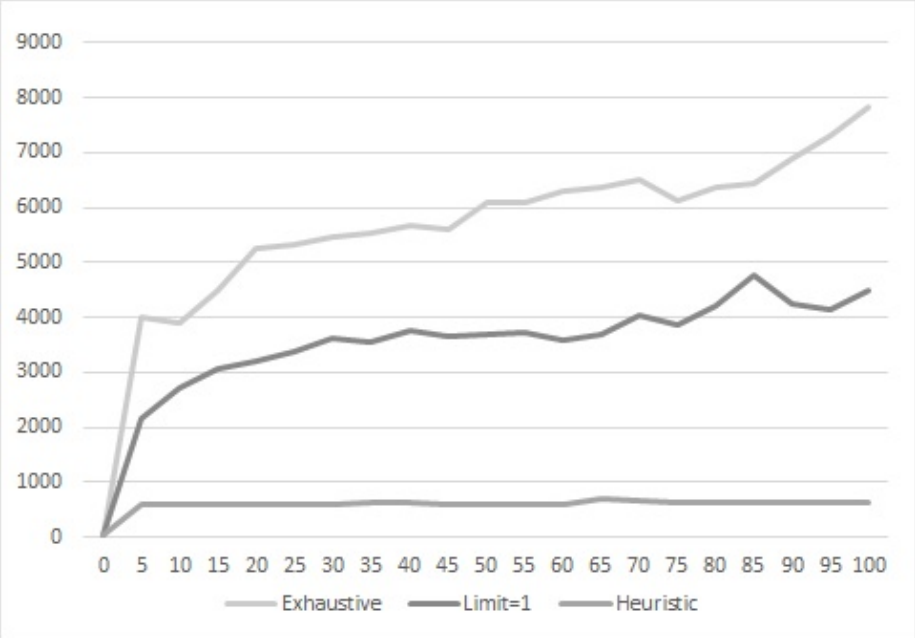


Figure 4: Time behavior of gFEED-ED^2 on changing object pixel densities

object pixel density of the image and the limited line scan is slightly affected by the object pixel density. The exhaustive line scan, however is clearly influenced by the object pixel density. Because the exhaustive line scan performs a scan of $\mathcal{O}(m+n)$ for each object pixel, the amount of object pixels have a bigger influence on the time than they do on the limited and heuristic line scan.

6.2 Euclidean Distance

gFEED[1] has also been tested on the datasets described above. The results of this tests are shown in Table 2.

Table 2: Timing results of the GDT on Euclidean distances

Dataset	Exh.	Limit=1	Heur.
Random dots	6977,8	4124,2	3936,3
Filled	3249,9	3507,7	3138,5
Open	3525,4	3700,5	3370,6
Filled/Open	3398,5	3550,8	3311,1
Lin. Gradient	3173,0	3273,3	3067,4
Rad. Gradient	3253,2	2982,3	2982,6
Lin. Grad. Noise	3699,4	3780,9	3645,7
Rad. Grad. Noise	3834,5	3717,5	3649,4
Objects	3704,8	1555,0	1664,0
Objects (r)oughened	38152,9	1575,3	1759,7
Objects (o)verlap-removal	27832,6	1593,8	1652,5
Objects o-r	25762,8	1464,0	1452,5

Average timing results (in ms) of three gFEED[1] algorithms.

This table shows that none of the three line scans does outperform the others. The exhaustive line scan is never the fastest algorithm and on the four black-and white images it is a lot slower than the limited and heuristic line scan. Because each object pixel has the same intensity, heuristic and limited line scans stop at the first object pixel found, while the exhaustive line scan continues. Therefore, using the exhaustive line scan is not recommended. The times of the limited and heuristic line scan are almost identical, this is caused by the amount of object pixels both algorithms feed. For example, on the 'Open' dataset, the limited line scan algorithm performs about 8000 more feeds on 100 images, than the heuristic line scan algorithm does. This small difference is also observed in other datasets.

7 Discussion

This paper introduced FEED-class algorithms for computation of the gDT. The gDT allows computation over gray-scale images. These FEED-class algorithms support GDTs on both ED and ED[2]. gFEED[2] computes the gDT exact and correct. Using limited or heuristic line scans, gFEED[2] shows a linear time complexity. LLT and PE however outperform gFEED[2].

For gFEED[1], the computation suffers from the same exactness problem as the naive quadratic algorithm does. However, we showed that gFEED[1] has a

$\mathcal{O}(m \times n)(m+n)$ time complexity instead of the naive $\mathcal{O}(m \times n)^2$ time complexity. This complexity is the equal for exhaustive, limited and heuristic line scans. In contrast to gFEED^2, gFEED^1 satisfies triangle inequality, this results in more intuitive results.

In the future, the speed of the gFEED might be increased by a better approximation of the bisection curve. Also, quadrant search is not included in this research, while it might be worthwhile for the computation in gFEED^2. New options to create bisection lines (i.e. by selecting random low intensity pixels) can also be researched.

References

[1] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics, and Image Processing*, 27(3):321–345, 1984.

[2] Enrique Coiras, Javier Santamaria, and Carlos Miravet. Hexadecagonal region growing. *Pattern Recognition Letters*, 19(12):1111–1117, 1998.

[3] Antonio Criminisi, Toby Sharp, Carsten Rother, and Patrick Pérez. Geodesic image and video editing. *ACM Trans. Graph.*, 29(5):134, 2010.

[4] Ricardo Fabbri, Luciano Da F Costa, Julio C Torelli, and Odemir M Bruno. 2d euclidean distance transform algorithms: A comparative survey. *ACM Computing Surveys (CSUR)*, 40(1):2, 2008.

[5] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. *Theory of Computing*, 8(19):415–428, 2012.

[6] Pedro F Felzenszwalb, Daniel P Huttenlocher, and Jon M Kleinberg. Fast algorithms for large-state-space hmms with applications to web usage analysis. *Advances in NIPS*, 16:409–416, 2004.

[7] Yves Lucet. New sequential exact euclidean distance transform algorithms based on convex analysis. *Image and Vision Computing*, 27(1):37–44, 2009.

[8] C. R. Maurer, Jr., R. Qi, and V. 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.

[9] T.E. Schouten and E.L. Van Den Broek. Fast exact euclidean distance (feed): A new class of adaptable distance transforms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(11):2159–2172, Nov 2014.

[10] Th E Schouten and EL van den Broek. Fast exact euclidean distance (feed) transformation. 2004.

[11] Frank Y Shih and Yi-Ta Wu. The efficient algorithms for achieving euclidean distance transformation. *Image Processing, IEEE Transactions on*, 13(8):1078–1091, 2004.

[12] Frank Y Shih and Yi-Ta Wu. Fast euclidean distance transformation in two scans using a 3× 3 neighborhood. *Computer Vision and Image Understanding*, 93(2):195–205, 2004.

[13] E. L. van den Broek and Th. E. Schouten. Distance transforms: Academics versus industry. *Recent Patents on Computer Science*, 4(1):1–15, 2011.

Appendices

A Exactness of Generalized Distance Transforms using Euclidean Distances

Suppose pixel p is fed by q with value $f(q) + d_{Eucl}(p,q)$. This is stored as a tuple $(n,m) = (f(q), d_{Eucl}(p,q)^2)$, so no value is rounded. Subsequently pixel q' starts feeding and feeds pixel p with $(n',m') = (f(q'), d_{Eucl}(p,q')^2)$ iff

$$\begin{aligned} f(q) + d_{Eucl}(p,q) &> f(q') + d_{Eucl}(p,q') \\ n + \sqrt{m} &> n' + \sqrt{m'}, \end{aligned} \tag{12}$$

which can be calculated as follows:

- if $n > n'$ and $m > m'$ then q' should update
- if $n' \geq n$ and $m' \geq m$ then q' should not update.

In all other cases, the calculation is less straightforward. The intuitive solution is to evaluate $\sqrt{m}$ and $\sqrt{m'}$, untill a difference occurs. Then, no two tuples A and B should be evaluated for which holds:

$$\begin{aligned} A &= (n,m) \\ B &= (n',m') \\ n' &\neq n \\ m &\neq m' \\ n + \sqrt{m} &= n' + \sqrt{m'}. \end{aligned} \tag{13}$$

If no such tuples A and B exist, the evaluation is always a finite process. If those tuples exist, they should be identified before evaluation.

Blömer [1] proves that for n, m, n' and m' being integer, is is possible to determine whether

$$\sqrt{n^2} + \sqrt{m} - \sqrt{n'^2} - \sqrt{m'} = 0 \tag{14}$$

with probability $\frac{1}{2}$. To my knowledge, no better algorithms exist. Hence, using current knowledge, both the standard GDT-ED and GDT-ED using FEED can only be approximated by evaluating the square root in ED-calculation.

References

[1] Blömer, Johannes. "Computing sums of radicals in polynomial time." Foundations of Computer Science, 1991. Proceedings., 32nd Annual Symposium on. IEEE, 1991.

B Implementation and experimentation

B.1 Introduction

This document gives an overview about what I have implemented and tested in the implementation. This implementation results are not important enough to be part of the paper. Instead, the work is documented here. In subsection 2, the implementations of gDT's using squared Euclidean Distances is discussed on four algorithms. In subsection 3, the implementations of gDT's using Euclidean Distances is discussed on two algorithms. subsection 4 discusses the generation of different types of test images.

B.2 Squared Euclidean Distance

B.2.1 Naive approach

The naive approach uses the algorithm based on the inverse definition for computing distance transform. For every object pixel, all background pixels are fed with the squared Euclidean Distance between them plus the value of the object pixel.

B.2.2 gFEED

gFEED is build on a n-dimensional grid framework, used for n-dimensional FEED. Since gFEED is only build for 2D images, the 2d-grid is changed to an 2-dimensional array before applying gFEED. After applying gFEED, the data is again changed to the original n-dimensional grid framework. Both operations cost $\mathcal{O}(n)$ time and are not included in timing experiments.

For gFEED on ED^2, three kinds of line scans are implemented:

1. **Exhaustive line scan:** This line scan checks every pixel from starting pixel $\bar{b}$ to the border and picks the object pixel $\bar{q}$ which reduces the bounding box the most. This pixel is identified as the pixel having the lowest value for

$$\frac{(km)^2 + (kn)^2 + f(\bar{q}) - f(\bar{b})}{k}. \tag{15}$$

2. **Limited line scan:** This line scan stops scanning when either the image border is found, or the limit l of found object pixels is exceeded. When an object pixel is scanned, its bisubsection line is compared to the currently best known bisubsection line, using Eq. 15. Only the best of the two bisubsection lines is stored for comparison.

3. **Heuristic limited line scan:** This line scan on through points $k(m, n)$ stops scanning when the either the image border is found, or the limit k exceeds a certain limit l. However, l is not a fixed variable, but is updated using the best known bisubsection line and the minimum value for $f(\bar{q}) = f_{min}(\bar{q})$. Given the best bisubsection line with a value x in Eq. 15, a better bisubsection line by $\bar{q} = k(m, n)$ has

$$\frac{(km)^2 + (kn)^2 + f(\bar{q}) - f(\bar{b})}{k} < x \qquad f(\bar{q}) \geq f_{min}(\bar{q}). \tag{16}$$

The best bisubsection line has

$$k \leq \frac{x + \sqrt{x * x - 4(f_{min}(\bar{q}) - f(\bar{b}))(m^2 + n^2)}}{2(m^2 + n^2)} \tag{17}$$

The maximum value of k is used to update the value of l.

All line-scans as described above are in directions (m, n) with $(|m|, |n|) \in \{(1, 1), (1, 2), (2, 1)\}$.

Initially, bounding boxes are created by scanning along the lines $(0, 1), (1, 0), (-1, 0)$ and $(0, -1)$. Using one of the line scanning methods. These lines create two horizontal and two vertical bisubsection lines, together this forms a bounding box. In FEED, only object pixels that do not have adjacent object pixels in horizontal and vertical direction, do have an initial bounding box covering a part of all quadrants. In gFEED, more object pixels do have this property. Therefore, an option is implemented, to create 4 bounding boxes. Each quadrant has its own bounding box, this results in more bounding boxes. However, this also increases the chance of one bounding box being reduced. Tests showed that this approach did not increase the speed of the algorithm, therefore it is not discussed in the paper.

A bounding box is reduced using the bisubsection lines. When using ED^2, bisubsection lines can lay behind the pixel. Therefore, the direction of reduction is important. This direction is determined using the scan line (m, n).

Verification of the implementation of the GDT-ED^2 using gFEED is done by comparison with the naive version of the GDT-ED^2.

B.2.3 PE

For testing and experimenting, another GDT-ED^2 algorithm is implemented. This algorithm is developed by Felzenzswalb and Huttenlocher. Its implementation is done using the pseudocode in [1]. Verification of this implementation is done by comparison with gFEED and by code comparison on the C++ version of the authors. The algorithm makes use of an horizontal and vertical pass and calculates the squared euclidean distance in one direction using Parabolic Envelopes (PE). We will refer to this algorithm as the PE-algorithm.

B.2.4 LLT

For testing and experimenting, the LLT algorithm as introduced by Lucet [2] is implemented. According to his paper, this algorithm would be faster than PE. LLT is based on the Linear-time Legendre Transform. When implementing LLT in C#; Lucet's paper, his implementation in SciLab and an implementation by Schouten and Van den Broek (S&B) were used as reference. My implementation is based on S&B. The division by two was eliminated and moved to the computation of the conjungate and the lower hull. However, S&Bs code did only support zero and infinity values, therefore, the code is changed to achieve this. The implementation is verified by comparison with PE.

B.3 Euclidean Distance

B.3.1 Naive approach

The naive approach uses the algorithm based on the inverse definition for computing distance transform. For every object pixel, all background pixels are fed with the Euclidean Distance between them plus the value of the object pixel.

B.3.2 gFEED

gFEED on ED is also build on the n-dimensional grid framework. Conversion to 2D arrays and back are not included in timing experiments. For gFEED on ED, three kinds of line scans are implemented:

- **Exhaustive line scan:** This line scan checks every pixel from starting pixel $\bar{b}$ to the border and picks the object pixel $\bar{q}$, having $f(\bar{q}) \leq f(\bar{b})$ which reduces the bounding box the most. The bounding box is reduced by a bisubsection line between $\bar{b}$ and $\bar{q}$. An image $\bar{q'}$ of $\bar{q}$ is created with $\bar{q'} = k'(m,n)$ and $f(\bar{q'}) = f(\bar{b})$. The best bisubsection line is created by $\bar{q}$ having the lowest value for k'.

- **Limited line scan:** This line scan checks a limited amount l of object pixels on the scanline having $f(\bar{q}) \leq f(\bar{b})$. If less than l object pixels exist on the scanline, the line scan stops at the image border.

- **Heuristic line scan:** The heuristic line scan is also limited by limit l. This limit is not fixed, but updated when a better bisubsection line is found. Given the minimum value f_{min} of an object pixel and pixel $\bar{q} = k(m,n)$ creating the current best bisubsection line, a better bisubsection line can only be drawn by pixels on the scanline not being farther than $f(\bar{q}) - f_{min}$ from $f(\bar{q})$. Therefore, l is updated to

$$\sqrt{\frac{(f(\bar{q}) - f_{min})^2}{m^2 + n^2}} + k. \tag{18}$$

  The scanline through points $k(m,n)$ is now limited to $k \leq l$.

All line-scans as described above are in directions (m,n) with $(|m|,|n|) \in \{(1,1),(1,2),(2,1)\}$.

Initially, bounding boxes are created by scanning along the lines $(0,1),(1,0),(-1,0)$ and $(0,-1)$. Using one of the line scanning methods. These lines create two horizontal and two vertical bisubsection lines, together this forms a bounding box. In FEED, only object pixels that do not have adjacent object pixels in horizontal and vertical direction, do have an initial bounding box covering a part of all quadrants. In gFEED, more object pixels do have this property. Therefore, an option is implemented, to create 4 bounding boxes. Each quadrant has its own bounding box, this results in more bounding boxes. However, this also increases the chance of one bounding box being reduced. Tests showed that this approach did not increase the speed of the algorithm, therefore it is not discussed in the paper.

Verification of the implementation of the GDT-ED using gFEED is done by comparison with the naive version of the GDT-ED.

B.4 Dataset Generation

Multiple types of datasets are generated, each dataset will be covered in a subsubsection below. Besides this generated datasets, some existing datasets will be used as well.

B.4.1 Random noise images

Datasets with random points with random intensities are generated by looping through the pixels and filling them with a random gray color with a give probability p. This results in images with a density of approximately p.

B.4.2 Random filled Objects

This Algorithm generates an image with random filled rectangles and ellipses. The size and amount of ellipses and rectangles is determined by user input. The rectangles and ellipses are fully filled and have a random orientation and gray-value.

B.4.3 Random open Objects

This Algorithm generates an image with random open rectangles and ellipses. The size and amount of ellipses and rectangles is determined by user input. The rectangles and ellipses have a random orientation and gray-value.

B.4.4 Random open and filled Objects

This Algorithm generates an image with random open and filled rectangles and ellipses. The size and amount of ellipses and rectangles is determined by user input. The rectangles and ellipses have a random orientation and gray-value.

B.4.5 Gradient objects

This algorithm generates an image with random filled rectangles and ellipses. The objects are filled with a random grayscale gradient (linear or radial). The gradient images can be generated with or without extra noise.

References

[1] P. F. Felzenszwalb and D. P. Huttenlocher. Distance transforms of sampled functions. Theory of Computing, 8(19):415-428, 2012.

[2] Yves Lucet. New sequential exact euclidean distance transform algorithms based on convex analysis. Image and Vision Computing, 27(1):37-44, 2009.