# Fast Exact Euclidean Distance (FEED) Transformation

Theo Schouten
Nijmegen Institute for
Computing and Information Science
University of Nijmegen
PO Box 9010, 6500GL Nijmegen
The Netherlands
ths@cs.kun.nl

Egon van den Broek
Nijmegen Institute for
Cognition and Information
University of Nijmegen
PO Box 9104, 6500HE Nijmegen
The Netherlands
e.vandenbroek@nici.kun.nl

## Abstract

*Fast Exact Euclidean Distance (FEED) transformation is introduced, starting from the inverse of the distance transformation. The prohibitive computational cost of a naive implementation of traditional Euclidean Distance Transformation, is tackled by three operations: restriction of both the number of object pixels and the number of background pixels taken in consideration and pre-computation of the Euclidean distance. Compared to the Shih and Liu 4-scan method the FEED algorithm is often faster and is less memory consuming.*

## 1. Introduction

A distance transformation [4] makes an image in which the value of each pixel is its distance to the set of object pixels $O$ in the original image:

$$D(p) = min\{dist(p,q), q \in O\} \quad (1)$$

Many algorithms to compute approximations of the Euclidean distance transformation (EDT) were proposed. Borgefors [1] proposed a chamfer distance transformation using two raster scans on the image, which produces a coarse approximation of the EDT. To get a result that is exact on most points but can produce small errors on some points, Danielsson [3] used four raster scans.

To obtain an exact EDT two step methods were proposed. Cuisenaire and Macq [2] first calculated an approximate EDT using ordered propagation by bucket sorting. It produces a result similar to Danielsson's. Then, this approximation is improved by using neighborhoods of increasing size. Shih and Liu [5] started with four scans on the image, producing a result similar to Danielsson's. A look-up table is then constructed containing all possible locations where no exact result was produced. Because during the scans the location of the closest object pixel is stored for each image pixel, the look-up table can be used to correct the errors. It is claimed that the number of error locations is small.

In contrast with these approaches, we have implemented the EDT starting directly from the definition in Equation 1. Or rather its inverse: each object pixel *feeds* its distance to all non-object pixels. This resulted in an exact but computationally less expensive algorithm for EDT: the Fast Exact Euclidean Distance *(FEED)* transformation.

Next, the principle of FEED and its methods to obtain a fast execution time are discussed. The third section describes the details of reducing the number of background pixels that must be updated, which is the largest contribution to obtaining the fast execution. In the last two sections results are presented and discussed.

## 2. Principle of FEED

In order to obtain an EDT, for each pixel $q$ in the set of object pixels ($O$) the Euclidean distance (ED) must be calculated to each background pixel $p$. The naive algorithm then becomes:

$$\begin{aligned} &initialize\ D(p) = if\ (p \in O)\ then\ 0,\ else\ \infty \\ &\quad foreach\ q \in O \\ &\quad\quad foreach\ p \notin O \\ &\quad\quad\quad update:\ D(p) = min(D(p), ED(q,p)) \end{aligned}$$

However, this algorithm is extremely time consuming, but can be speeded up by:

- restricting the number of object pixels $q$ that have to be considered
- pre-computation of $ED(q,p)$
- restricting the number of background pixels $p$ that have to be updated for each considered object pixel $q$

Only the "border" pixels $B$ of an object have to be considered. A border pixel $B$ is defined as an object pixel with at least one of its four 4-connected pixels in the background. It can then be easily proved that the minimal distance from any background pixel to an object, is the distance from that background pixel to a border pixel $B$ of that object.

Since the ED is translation invariant, the EDs can be precomputed and stored in a matrix $M(x,y) = ED((x,y),0)$. $ED(q,p)$ is then taken as $ED(q-p,0)$ from the matrix. In principle the size of the matrix is twice the size of the image in each dimension. If the property $ED((x,y),0) = ED((|x|,|y|),0)$ is used in the updating of $D(p)$, only the positive quadrant of $M$ is needed. Hence, the size of the matrix becomes equal to the image size. Its calculation can be speeded up using the the fact that ED is symmetric: $ED((x,y),0) = ED((y,x),0)$.

If an upper limit of the maximum value of $D(p)$ in an image is known a priori, the size of $M$ can be decreased to just contain that upper limit. This would increase the speed of the algorithm (e.g., in a situation where fixed objects are present). The size of $M$ can also be decreased if one is only interested in distances up to a certain maximum. For example, in a robot navigation problem where only small distances give navigation limitations.

Due to the definition of $D(p)$ the matrix $M$ can be filled with any non-decreasing function $f$ of ED: $f(D(p)) = min(f(D(p)), f(ED(q,p)))$. For instance, the square of ED allowing the use of an integer matrix $M$ in the calculation. Alternately, one can truncate the ED to integer values in $M$ when it is stored in such format in the final $D(p)$.

The number of background pixels that have to be updated can be limited: only those that have an equal or smaller distance to the border pixel $B$ than to an object pixel $p$ (see Figure 1). The equation of the bisection line $b$ with the origin of the coordinate system placed at $B$ is: $p_y y + p_x x = (p_x^2 + p_y^2)/2$.

Regarding the speed of the algorithm the problem is that not too much time should be spend on searching for other object pixels, on the administration of the bisection lines, or on determining which pixels to update. That is because the update operation is simply a test followed by one assignment.

## 3. Reducing the number of pixels to update

The search for object pixels $p$ is done on lines through the current border pixel ($B$) with certain $m = p_y/p_x$ ratio's. Define $m = m_1/m_2$ with $m_1$ and $m_2$ the minimal integers, then the equation of the bisection becomes: $2(m_1 m_2 y + m_2^2 x) = (m_1^2 + m_2^2)p_x$. This is of the form: $m_a y + m_b x = m_c p_x$ with $m_a$, $m_b$ and $m_c$ integers that depend only on $m$. For each quadrant for each $m$ only the object pixel closest to $B$ is relevant, searching along the line can be stopped as soon as one is found. The resulting bisection line $b$ is then identified with the quadrant number, with $m$, and with $p_x$.

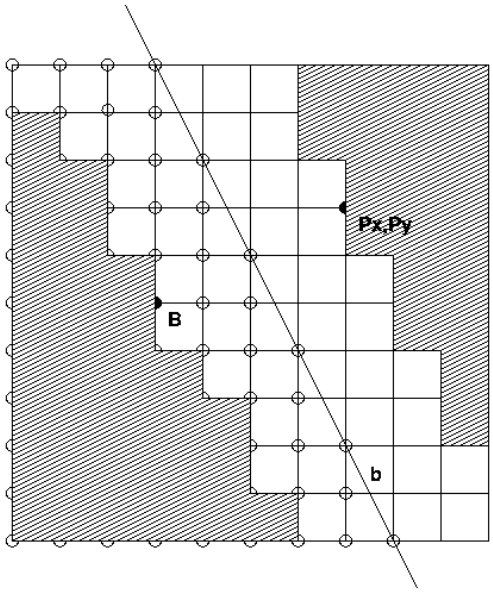


**Figure 1. Principle of limiting the number of background pixels to update. Only the pixels on and to the left of the bisection line $b$ have to be updated. $B$ is the border pixel under consideration, $p$ is an object pixel.**

To keep track of the update area, the maximum x and y values of each quadrant are updated (see Figure 2). Only pixels inside each square need to be updated, but not all of them. A bisection line $b$ in a quadrant might update these maximum values in this and two neighboring quadrants, as is indicated with the open arrows in Figure 2. For example: $max_{y1} = min(max_{y1}, m_c p_x/m_a)$. The intersection point of two bisection lines in different quadrants might also give a new maximum value, as indicated with the two closed arrows in quadrant 1. The maximum values can be calculated using integers, but it takes time.

The maximum values also determine the distance to search along each search line. For example, for a search line in quadrant 1 at least one of the points $(max_{x1}, max_{y1})$, $(0, max_{y2})$, and $(max_{x4}, 0)$ must be on or to the right of the bisection line. This gives a maximum value for $p_x$ of $max(m_a max_{y1} + m_b max_{x1}, m_a max_{y2}, m_b max_{x4})/m_c$.

Bisection lines closer to the origin $B$ have a larger effect than lines further away. Searching in circular patterns the closest lines are found first, thus less points are checked than using a radial search. But it requires a more time consuming checking of reaching a maximum value than when using radial search lines.

Since in general a number of object points from the same

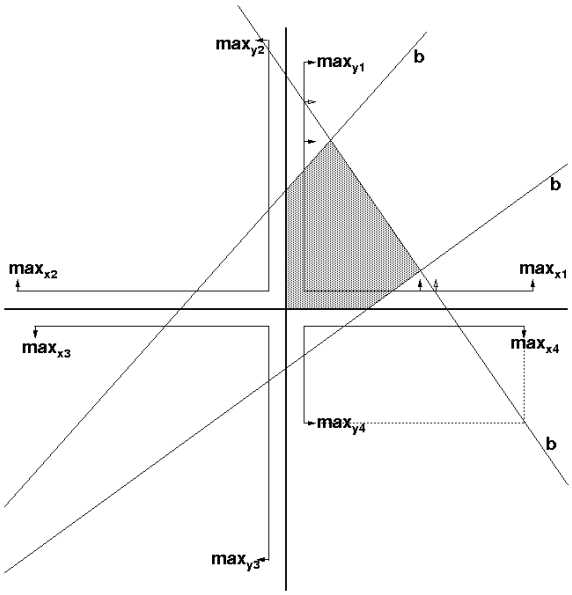


**Figure 2. Bookkeeping of the sizes (the "max") of each quadrant. Updating process: On each scan line the bisection lines $b$ determine the range of pixels to update.**

object are close to $B$, the radial search is splitted. In a small area around $B$ all the points are checked before checking a number of radial lines further out. To increase the speed, not all points on them are checked but a certain stepping is used. If the remaining search area, calculated as $\sum_{i=1}^{4} max_{xi} max_{yi}$, is small enough, searching is stopped.

The final selection of pixels to update is made in the update process. For each scan line in a quadrant, the maximum x and y values of the quadrant and the found bisection lines in that and neighboring quadrants, determine start and end points, as is indicated with the shaded area in Figure 2.

In addition, some further speedups are implemented, using information saved from a border point for next border points. By searching for border pixels along horizontal scan lines, the search for object pixels along the $m = 0$ line can be combined with it. For searching in the vertical direction a binning of the image in the vertical direction is used, which is done combined with the initialization of $D(p)$. Pixels exactly on a bisection line, have only to be updated once. This is done in quadrants 1 and 2, by simply decreasing $m_c p_x$ by 1 for quadrants 3 and 4.

## 4. Results

For our FEED method test lines with $m$'s of 1/4, 1/3, 1/2, 2/3, 3/4, 1, 4/3, 3/2, 2, 3, and 4 and both the vertical and horizontal lines were chosen. The intersection points of bisection lines were only calculated for bisection lines with the same angle. The small search area around each border pixel was set to a square of 9 by 9 pixels and for the larger area a stepping of 8 pixels was used. The search area limit was set to 1000 pixels. For the bin size 1% of the image height was choosen. All the above settings were based on experiments using a large number of images.

Our implementation was checked using small images for which the correct ED can easily be determined. That include the images given in reference [2] and [5] that gives errors in ED in the first steps of those methods. Further the results on various test images using the simple algorithm in Section 2 without any speed optimization, were used to check the implementation of the optimizations.

As noted by Cuisenaire and Macq [2] comparing the complexity and computational costs of EDT algorithms is a complex task. Currently, the 4-scan method of Shih and Liu [5] is one of the fastest EDT methods. Therefore, we have chosen to compare FEED with this method, except for their look-up table correction. The execution time of the 4-scan method increases proportional to the number of pixels in the image. In images of a given size, the time will depend in some complex way on the distribution of object pixels and no equation was given for that. Similarly, it is not possible to determine the exact computational complexity of FEED. However, it should be noted that FEED uses two scans over the image, one for initialization and one for finding the border pixels.

We started our comparison with two test images suggested by Cuisenaire and Macq [2]: a circle nearly covering the whole image and a line under an angle of $20°$. Large images of 4096 by 4096 pixels were used to get correct time measurements. Results of the two images and their negatives on a SUN machine are given in Table 1. In the column "reduced $M$", the results are given for FEED with the size of the matrix $M$ set to the maximal occurring distance in the image. All FEED results are faster up to a maximum factor of 3 and both methods show a large variation in execution time (see Table 1) .

Since we were developing FEED for a robot navigation problem, we generated images containing objects: "objects" and "test-obj". Their features were designed in such a way that errors in the implementation were likely to become present. Other images generated were random blob images. They consisted of respectively 1%, 5%, and 10% out of border pixels. The latter is indicated by the number behind their name (see Table 1).

With these images FEED proved, once more, to be faster. In addition, again, both methods show a large variation in execution time. Similar results were found for other generated blob images that are not shown in Table 1.

Shih and Liu [5] argue that the number of pixels with a wrong ED after their 4 scans over the image is less than

| image | Shih & Liu | FEED | reduced $M$ |
|---|---|---|---|
| circle | 22.6 s | 16.9 s | 14.9 s |
| line | 29.7 s | 13.5 s | 11.9 s |
| neg circle | 10.7 s | 5.9 s | 3.4 s |
| neg line | 5.4 s | 4.6 s | 2.1 s |
| test-obj | 23.3 s | 8.7 s | 5.2 s |
| objects | 24.5 s | 9.6 s | 5.6 s |
| neg test-obj | 8.7 s | 5.9 s | 3.4 s |
| neg objects | 9.9 s | 6.7 s | 4.3 s |
| blob01 | 23.7 s | 10.4 s | 6.4 s |
| blob05 | 20.6 s | 14.1 s | 10.2 s |
| blob10 | 19.1 s | 18.8 s | 15.4 s |

**Table 1. Timing results for the Shih and Liu 4-scan method, for FEED with a full matrix $M$ and for FEED with matrix $M$ reduced to the a priori known maximum distance in the image.**

1%. We found for the circle image that 8.3% and for the line image that 68.8% of the pixels were wrong. For the other images tested, the percentage of pixels with a wrong ED ranged from 0.1 to 3.1%. Hence, since we have not implemented Shih and Liu's correction of wrong EDs, the speed advantage of FEED is even larger than indicated in the table.

Last, we considered images on which FEED would not perform well: random dot images. In such images, for each pixel it is decided randomly, independent from the other pixels, whether the pixel becomes white or black. For 10% object pixels FEED is still a factor 1.5 faster. When the percentage is increased the 4-scan method becomes faster than FEED. The speed difference is the highest at 80% object pixels where FEED is a factor of 6 slower. These results are partly explained by the fact that the effect of reducing the size of $M$, for those images, is small.

Note that the methods to obtain a fast execution were specially adapted to object like images. So far we did not consider methods to speed up the execution of random dot like images. Thus, the current FEED implementation is regarding execution time suited for object like images, but not for random dot like images.

Regarding space requirements, both FEED and the 4-scan algorithm use an integer matrix of a size equal to the image size to store the EDs. The 4-scan method uses two of such matrices to store the source mapping additionally. Hence, FEED uses much less space than the 4-scan method.

All images used for testing FEED can be found on *http://www.cs.kun.nl/∼ths/feed*

## 5. Discussion

We have developed a Fast Exact Euclidean Distance (FEED) transformation, starting from the inverse of the distance transformation: each object pixel feeds its distance to all background pixels. The prohibitive computational cost of this naive implementation of traditional ED transformations, is tackled by three operations: restriction of both the number of object pixels and the number of background pixels taken into consideration as well as the pre-computation of the ED.

We tested FEED on many images and compared the results with the results from the Shih and Liu 4-scan method. For images containing objects FEED is faster up to a maximum factor of 3 and even 4 if the maximum occurring distance is provided as input to FEED. These kind of images occur, for example, in applications like robot navigation. For images containing random pixels FEED can be slower than the 4-scan method. These kind of images can, for instance, occur in classification applications.

FEED is less memory consuming than the 4-scan method. For an input image of $n$ pixels, FEED uses $n$ integers to store the EDs. In contrast, the 4-scan method uses $3n$ integers to store the EDs and additional information, for later correction of wrong EDs.

Further research will focus on boosting FEED, especially for random dot images. We will further look into using FEED for video sequences. In the case that fixed objects are present, their influence on the $D(p)$ matrix can be precalculated and the changing objects in each image can then be added to the $D(p)$ of that image. Last, note that FEED is essentially parallel so that parallel implementations can be developed if the need arises.

So, with FEED no approximations of ED transformations are needed due to its computational burden, but both Fast and Exact ED transformations can be done on images containing objects. With that a new image processing algorithm is launched important for many applications in image analysis.

## References


[1] G. Borgefors. Distance transformations in digital images. *Computer Vision, Graphics and Image Processing*, 34:344–371, 1986.

[2] O. Cuisenaire and B. Macq. Fast euclidean transformation by propagation using multiple neighborhoods. *Computer Vision and Image Understanding*, 76:163–172, 1999.

[3] P. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[4] A. Rosenfeld and J. Pfaltz. Distance functions on digital pictures. *Pattern Recognition*, 1:33–61, 1968.

[5] F. Y. Shih and J. J. Liu. Size-invariant four-scan euclidean distance transformation. *Pattern Recognition*, 31:1761–1766, 1998.