An Efficient Euclidean Distance Transform

Donald G Bailey

Institute of Information Sciences and Technology,
Massey University, Palmerston North, New Zealand
D.G.Bailey@massey.ac.nz

Abstract. Within image analysis the distance transform has many applications. The distance transform measures the distance of each object point from the nearest boundary. For ease of computation, a commonly used approximate algorithm is the chamfer distance transform. This paper presents an efficient linear-time algorithm for calculating the true Euclidean distance-squared of each point from the nearest boundary. It works by performing a 1D distance transform on each row of the image, and then combines the results in each column. It is shown that the Euclidean distance squared transform requires fewer computations than the commonly used 5x5 chamfer transform.

1 Introduction

Many image analysis applications require the measurement of objects, the components of objects or the relationship between objects. One technique that may be used in a wide variety of applications is the distance transform or Euclidean distance map [1,2]. Let the pixels within a two-dimensional digital image $I(x, y)$ be divided into two classes – object pixels and background pixels.

$$I(x, y) \in \{Ob, Bg\} \tag{1}$$

The distance transform of this image, $I_d(x, y)$ then labels each object pixel of this binary image with the distance between that pixel and the nearest background pixel. Mathematically,

$$I_d(x, y) = \begin{cases} 0 & I(x, y) \in \{Bg\} \\ \min\left(\left\|x - x_0, y - y_0\right\|, \forall I(x_0, y_0) \in Bg\right) & I(x, y) \in \{Ob\} \end{cases} \tag{2}$$

where $\|x, y\|$ is some two-dimensional distance metric. Different distance metrics result in different distance transformations. From a measurement perspective, the Euclidean distance is the most useful because it corresponds to the way objects are measured in the real world. The Euclidean distance metric uses the L_2 norm and is defined as

$$\|x, y\|_{L_2} = \sqrt{x^2 + y^2} \tag{3}$$

R. Klette and J. Žunić (Eds.): IWCIA 2004, LNCS 3322, pp. 394–408, 2004.

This metric is isotropic in that distances measured are independent of object orientation, subject of course to the limitation that the object boundary is digital, and therefore in discrete locations. The major limitation of the Euclidean metric, however is that it is not easy to calculate efficiently for complex shapes. Therefore several approximations have been developed that are simpler to calculate for two-dimensional digital images using a rectangular coordinate system. The first of these is the city block, or Manhattan metric, which uses the L_1 norm

$$\|x, y\|_{L_1} = |x| + |y| \tag{4}$$

where the distance is measured by the number of horizontal and vertical steps required to traverse (x,y). If each pixel is considered a node on a graph with each node connected to its 4 nearest neighbours, the city block metric therefore measures the distance as the minimum number of 4-connected nodes that must be passed through. Diagonal distances are over-estimated by this metric because a diagonal connection counts as 2 steps, rather than $\sqrt{2}$.

Another measure commonly used is the chessboard metric, using the L_∞ norm

$$\|x, y\|_{L_\infty} = \max(|x|, |y|) \tag{5}$$

which measures the number of steps required by a king on a chess board to traverse (x,y). The chessboard metric considers each pixel to be connected to its 8 nearest neighbours, and measures the distance as the minimum number of 8-connected nodes that must be passed through. Diagonal distances are under-estimated by this metric as a diagonal connection counts as only 1 step.

A wide range of other metrics have been proposed that aim to approximate the Euclidean distance while retaining the simplicity of calculation of the city block and chessboard metrics. Perhaps the simplest of these is to simply average the city block and chessboard distance maps:

$$\begin{aligned} \|x, y\|_{Hybrid} &= \tfrac{1}{2}\left(|x| + |y| + \max(|x|, |y|)\right) \\ &= \max(|x|, |y|) + \tfrac{1}{2}\min(|x|, |y|) \end{aligned} \tag{6}$$

Fig. 1 graphically compares these different metrics in measuring the distance from a point in the centre of an image. The anisotropy of the non-Euclidean distance measures is clearly visible.

1.1 Morphological Approach – Grassfire Transform

Calculation of the distance transform directly using Eq. (2) is impractical because it involves measuring the distance between every object pixel and every background pixel. The time required would be proportional to the number of pixels in the image squared. Therefore more efficient algorithms have been developed to reduce the computational complexity.

Intuitively, the simplest approach to calculate the distance transform is to iteratively label each pixel starting from the edges of the object. The so-called grassfire transform imagines that a fire is started at each of the edge pixels which burns with constant velocity. An object pixel's distance from the boundary is therefore given by the time it takes the fire to reach that pixel. The grassfire transform is initialised by labelling all of the background pixels as 0. In iteration i, each unlabelled object pixel that is adjacent to (using 4-connections for the city block metric or 8-connections for the chessboard metric) a pixel labelled i-1 is labelled i. The iterations continue until all of the pixels have been labelled.

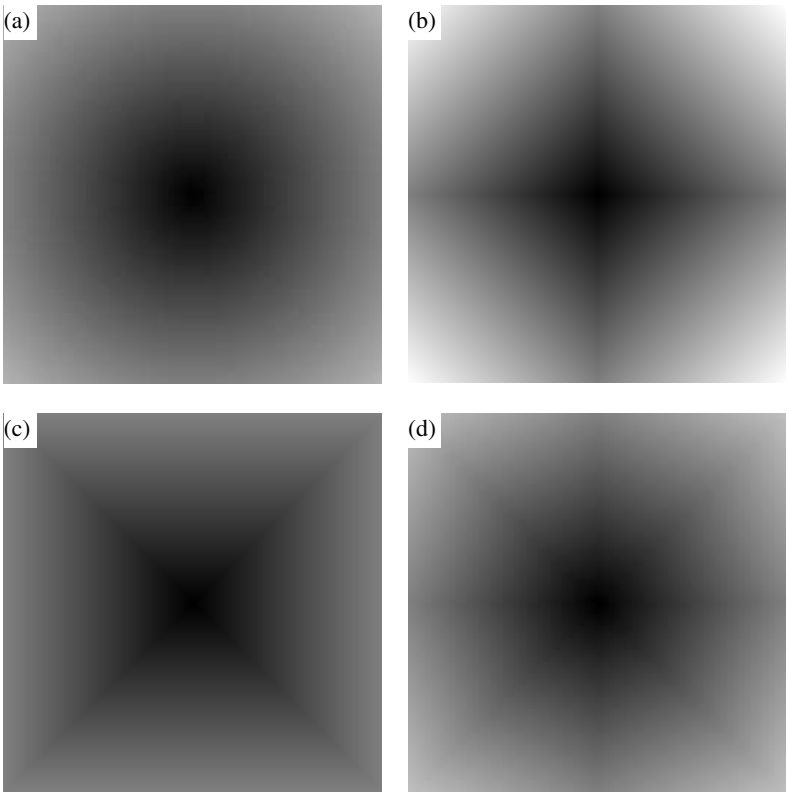

Fig. 1. Four commonly used distance metrics – measuring the distance from the centre of the image: (a) Euclidean metric, Eq. (3); (b) city block metric, Eq. (4); (c) chessboard metric, Eq. (5); (d) a hybrid metric, Eq. (6)

This iterative approach is like peeling the layers of an onion. This may be achieved by using a morphological filter to erode the object by one layer at each iteration. The shape of the structuring element (see Fig. 2) determines which distance metric is being applied. Each pixel is then labelled by the iteration number at which it was eroded from the original image. The hybrid distance metric of Eq. (6) may be achieved by alternating the cross and square structure elements at successive iterations [2].

The major limitation of using such small structuring elements is that many iterations are required to label large objects. Also, the hybrid metric provides only a crude approximation of the Euclidean distance. Both of these limitations may be overcome by greyscale morphology with a conical structuring element. In general, larger structuring elements require fewer iterations and the final result more closely approximates the Euclidean distance. However, the cost is that larger structuring elements are more computationally intensive at each iteration. For this reason, much research has gone into ways of decomposing the conical structuring element to reduce the computational burden (see for example [3-5]).

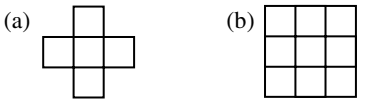

Fig. 2. Structure elements for: (a) city block erosion; and (b) chessboard erosion

1.2 Two Pass Algorithms – Chamfer Distance Transform

The iterations required by successive use of morphological filters may be removed by making the observation that successive layers will be adjacent. Therefore the distance may be calculated by propagating the distances from adjacent pixels. This approach requires only two passes through the image, one from the top left corner to the bottom right corner and the second from the bottom right back through the image to the top left corner. These two passes propagate the distances from the top and left edges of the object, and from the bottom and right edges of the object respectively. Each pass uses only values that have already been calculated.

If using a 3x3 window, the first pass propagates the distance from the 3 pixels above, and the one pixel to the left of the current pixel, adding an increment that depends on whether the pixel is 4- or 8-connected. Background pixels are assigned a distance of 0.

$$I_d(x,y) = \min\begin{pmatrix} I_d(x-1,y-1)+b, & I_d(x,y-1)+a, & I_d(x+1,y-1)+b, \\ I_d(x-1,y)+a & & \end{pmatrix} \tag{7}$$

The second pass propagates the distance from the 3 pixels below and the one pixel to the right of the current pixel. The second pass only replaces the distance calculated in the first pass if it is smaller, which will be the case if the pixel is closer to the bottom or right edges of the object.

$$I_d(x,y) = \min\begin{pmatrix} & I_d(x,y), & I_d(x+1,y)+a, \\ I_d(x-1,y+1)+b, & I_d(x,y+1)+a, & I_d(x+1,y+1)+b \end{pmatrix} \tag{8}$$

Different increments, a and b, will result in different distance metrics. The city block distance is given with a=1 and b=2; the chessboard distance with a=b=1; and the hybrid distance of Eq. (6) is given with a=1 and b=1.5, or equivalently with a=2 and b=3 (to maintain integer arithmetic), and dividing the result by 2. A better ap-

proximation to the Euclidean distance may be obtained by using the integer weights a=3 and b=4, and dividing the result by 4, although this still results in an octagonal pattern similar to that seen in Fig. 1 (d).

A more accurate distance measure may be obtained by optimising the increments, or by using a larger window size [6]. A larger window size compares more terms (4 for a 3x3 window, 8 for a 5x5 window, and 16 for a 7x7 window - see Fig. 3), and provides a more accurate estimate of distances that are off-diagonal. A 5x5 window provides a reasonable compromise between computational complexity and approximation accuracy, and is commonly used when a closer approximation to the true Euclidean distance is required. As the number of operations is fixed for each pixel, the time required to execute the chamfer distance algorithms is proportional to the number of pixels in the image.

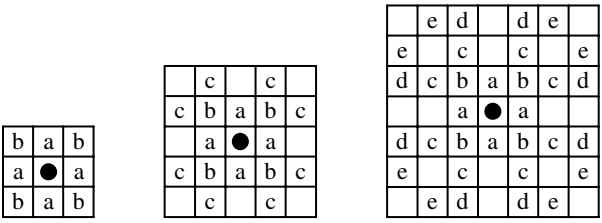

Fig. 3. The location of increments within 3x3, 5x5 and 7x7 windows. The blank spaces do not need to be tested because they are multiples of smaller increments

1.3 Vector Propagation

The two-pass chamfer distance algorithm may be adapted to measure the Euclidean distance by propagating vectors instead of the scalar distance [7-9]. The basic approach remains the same – as the window is scanned through the image, the distance is calculated by minimising an incremental distance from its neighbours. Measuring the Euclidean distance requires a square-root operation. However, if the minimum distance is selected, then the distance squared will also be minimised. This reduces the number of expensive square root operations that are actually needed. In many applications, the distance squared transform is suitable, avoiding square roots altogether.

Whereas the chamfer distance only requires an image of scalars, measurement of the Euclidean distance requires an intermediate image of vectors with x and y offsets. Background pixels are assigned a vector of (0,0). The minimum distance is calculated by propagating the vector components of each of the neighbours that have already been calculated. A similar operation is performed for the second pass, from the bottom right back up the image to the top left.

Consider an isolated background pixel (a single pixel hole in an “infinite” object). The first pass will propagate the correct distances downwards in the image as illustrated in Fig. 4. The pixels in the lower right quadrant have 3 redundant paths from adjacent pixels. The redundancy is less in the lower left octant because the pixel immediately to the right of the current pixel has not been processed yet, so has an unknown distance from the background.

The second pass is more revealing. The top left quadrant is fully redundant. In the top right quadrant, there are no direct right propagations because the pixel immediately to

the left of the current pixel has not yet been processed. The bottom left octant has no redundancy. The propagation path to any pixels in this region follows the bottom left diagonal in the first pass, and then left from that in the second pass.

This lack of redundancy means that every pixel on the propagation path must be closer to the original background point than any other background pixel. If not, for example if any of the diagonal pixels is closer to another background pixel, then the pixels within this region will have the incorrect minimum distance. Fig. 5 shows a construction where this will be the case, and there is an error in the derived distance.

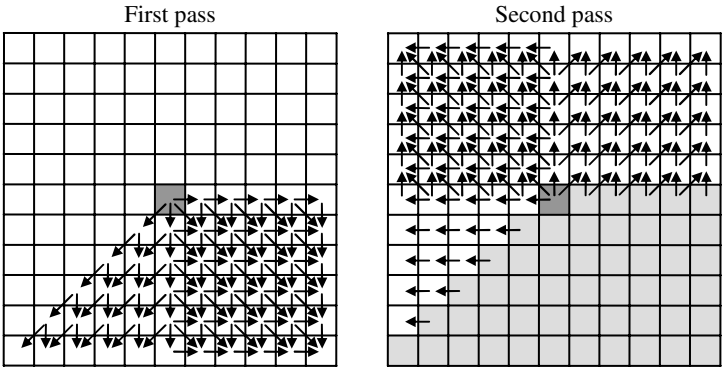

Fig. 4. Propagation of distances from a single background pixel within the two passes of the algorithm. Arrows show where the minimum comes from. Where there are multiple arrows entering a pixel, all of the paths equal the minimum

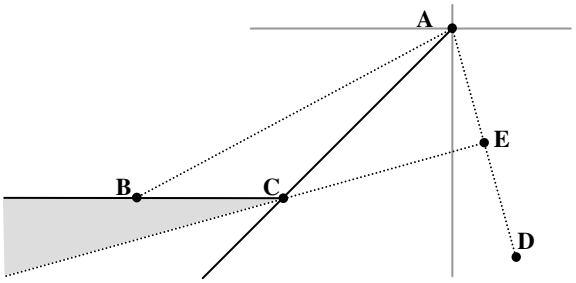

Fig. 5. Construction illustrating regions which will have an incorrect distance calculated. **A** and **D** are background pixels. Line **CE** is the perpendicular bisector of line **AD**, and consists of the points which are equidistant from **A** and **D**; points above this line are closer to **A** and points below this line are closer to **D**. The propagation of minimum distance from **A** to **B** follows the lower-left diagonal from **A** to **C**, then left to **B** (see Fig. 4). Pixels below **BC** that are above **CE** (shaded region) will be using diagonal pixels that have been labeled as being closer to **D** and so will have incorrect distances

These errors may be corrected by allowing a more direct path between point **A** and **B**. This requires a third pass through the image to provide the missing diagonal connections in this octant. It is also necessary to include the right to left propagation to correct

any errors resulting from the propagation of incorrect distances in this direction. To accommodate both these propagations, it is necessary for the third pass to proceed from the top right corner to the bottom left corner, traversing right to left along each row of pixels. The redundancy added by this pass will enable the correct distances to be obtained when there are two background points. (This source of errors is overlooked in Shih and Fu [9]).

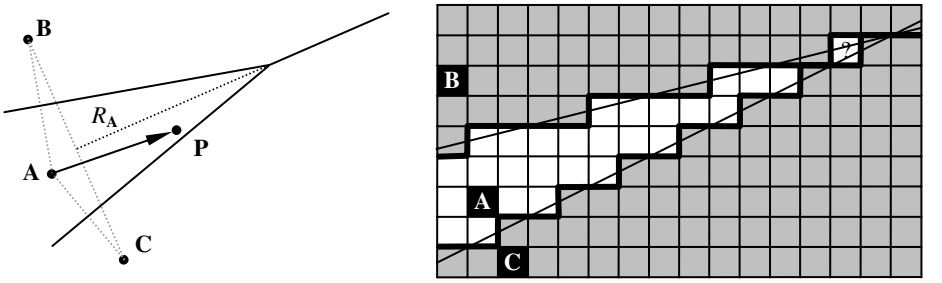

Fig. 6. Construction where there are three background pixels. The correct distances will be measured as long as there is a propagation path completely within the region associated with each background pixel. In continuous images, this will always be the case. With digital images, however, the boundaries are not straight lines, but jagged digital lines. This is illustrated in the example on the right, where there is an isolated pixel failing this criterion

Now consider the case where there are three background points, as illustrated in Fig. 6. The perpendicular bisectors between each pair of points govern the boundaries between the regions made up of points closest to each of the three background pixels. Consider pixel **A**, the central point of the three, and its associated region, $R_{\mathbf{A}}$. If there is a connected propagation path completely within the region associated with that point, then each pixel will have the correct distance. This is because each pixel along the path will be propagating the correct distance.

In the general case, when there are many background pixels within an image, the region $R_{\mathbf{A}}$ consisting of all of the point closest to a given background pixel, **A**, may be constructed as follows. The perpendicular bisector of the line between **A** and another background pixel **B** consists of all points that are equidistant to both **A** and **B**. All points on the **A** side of the bisector are closer to **A**. A point is in region $R_{\mathbf{A}}$ only if it is on the **A** side of all such bisectors. Therefore, $R_{\mathbf{A}}$ consists of the intersection of all such regions:

$$R_{\mathbf{A}} = \bigcap \left\{ \mathbf{P} \middle\| \mathbf{P}-\mathbf{A} \right\| \le \left\| \mathbf{P}-\mathbf{B} \right\|, \forall \mathbf{P} \in Ob, \mathbf{B} \in Bg, \mathbf{A} \ne \mathbf{B} \right\} \tag{9}$$

The division of an image into regions in this manner is called the Voronoi diagram. The Voronoi diagram effectively associates each point within an image with the nearest feature (or background) point. Therefore obtaining the distance transform from the Voronoi diagram is a relatively simple matter [10,11].

From a vector propagation standpoint, since the Voronoi region R_A is convex, the line segment between **A** and any point within $R_{\mathbf{A}}$ will lie completely within $R_{\mathbf{A}}$. As a result, provided distances may propagate along this line segment, the correct distance will be obtained for every point in $R_{\mathbf{A}}$ and by generalization, any object point.

For continuous images, this will always be the case. However, for digital images, the boundaries of R_A are not continuous lines, but are digital lines, and are distorted by the pixel grid. When two digital bisectors approach at an acute angle, as shown in the example in Fig. 6, there may be an isolated pixel, or short string of pixels that are not 8-connected with the rest of the region [12]. Consequently, there will not be a continuous 8-connected path between such groups and the nearest background pixel for the distances to propagate along. These groups will therefore not have the correct minimum distances assigned to them. It can be shown that using a small local window cannot prevent such errors [12].

1.4 Boundary Propagation

Another class of techniques combines the idea of the grassfire transform with the propagation approach described in the previous section. These methods maintain a list of boundary pixels, and propagate these in a non-raster fashion [12-14]. Redundant comparisons may be avoided by only testing based on the direction of the nearest boundary pixel [14]. Errors such as that shown in Fig. 6 may be avoided by propagating vectors past the maximum until the difference exceeds 1 pixel [13]. While this extended propagation overcomes these errors, if care is not taken these additional propagations can result in large numbers of unnecessary comparisons [12].

1.5 Independent Scanning of *x* and *y*

The definition of Euclidean distance in Eq. (3) leads to a different class of algorithms. From Pythagoras' theorem, the distance squared to a background pixel can be determined by considering the x and y components separately. Therefore it is possible to independently consider the rows and columns. The first step looks along each row to determine the distance of each object point from the nearest boundary point on that row. This requires two scans, from left to right and right to left to measure the distances from the left and right edges of the object respectively. The second step then considers each column, and for each pixel in that column determines the closest background point by examining only the row distances in that column:

$$I_d^2(x, y) = \min_n \left(I(x, y_n)^2 + (y - y_n)^2 \right) \tag{10}$$

Thus the search has been reduced from two dimensions in Eq. (2) to one dimension. The search can be accomplished with a scan down and up the column propagating the row distances and selecting the global minima at each pixel [15]. Unfortunately, as applied, this algorithm requires that multiple row points be propagated simultaneously. The effect is that in the worst case the algorithm as described is not linear in the number of pixels (as are the chamfer and vector propagation algorithms).

2 Linear Time Independent Scanning

The key to making an independent scanning algorithm operate in linear time is to determine in advance exactly which pixels in a column that a particular row will in-

fluence. This information may be obtained by constructing a partial Voronoi diagram for each column.

2.1 Row Scanning

The first step operates on each row independently. It consists of two passes – from left to right and then right to left. The left to right pass determines the distance to the left boundary of an object

$$I(x,y)=\begin{cases} I(x-1,y)+1 & I(x,y)\in Ob \\ 0 & I(x,y)\in Bg \end{cases} \tag{11}$$

If the pixel on the edge of the image is an object pixel, its distance is set to ∞. The right to left pass replaces this with the distance to the right boundary if it is shorter:

$$I(x,y)=\begin{cases} \min(I(x,y),I(x+1,y)+1) & I(x,y)\in Ob \\ 0 & I(x,y)\in Bg \end{cases} \tag{12}$$

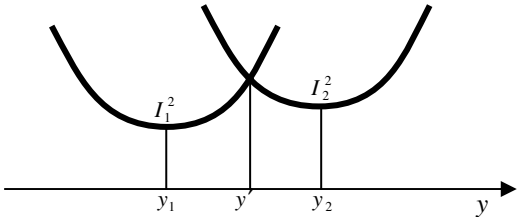

Fig. 7. The distance squared along a column, showing the regions of influence of two background points

2.2 Column Scanning

Consider an image with two background pixels at $I(x_1,y_1)$ and $I(x_2,y_2)$, with $y_1<y_2$. Let I_1 and I_2 be the corresponding minimum row distances in column x. The distance squared function in column x is illustrated in Fig. 7. The column is split into two with part of the column coming under the influence of (x_1,y_1) and part coming under the influence of (x_2,y_2). The boundary between the two regions is given from the intersection of the two parabola:

$$I_1^2+(y'-y_1)^2=I_2^2+(y'-y_2)^2 \tag{13}$$

Solving this for the position of the intersection gives:

$$y'=y_2+\frac{I_2^2-I_1^2-(y_2-y_1)^2}{2(y_2-y_1)} \tag{14}$$

Note that there will always be exactly 1 intersection point, corresponding to where the perpendicular bisector between $I(x_1,y_1)$ and $I(x_2,y_2)$ intersects column x, although the bisector may not necessarily be between y_1 and y_2. As the distance is only evaluated for integer values of y, it is not necessary to know the precise location of the intersection, only which two pixels it falls between. This means that integer division may be used, and the remainder or fractional part discarded. If the numerator is positive, the number calculated is the last pixel under the influence of y_1. If negative, it is the first pixel under the influence of y_2.

Assume that the image is being scanned in the increasing y direction. Now consider adding a third background point $I(x_3,y_3)$, where $y_2<y_3$ with intersection between parabolas 2 and 3 at y''. If $y' < y''$ then there are three regions of influence, corresponding to the sets of points nearest to each of the background pixel. However if $y'' < y'$ then background point 2 has no influence in column x because its parabola will be greater than the minimum of parabolas 1 and 3 at every point. The boundary between parabolas 1 and 3 may then be found from Eq. (14).

Extending this search to N points would require N^2 tests in the worst case. However, by making use of the fact that the points are ordered, and scanning in only one direction at a time, the number of tests may be reduced to N.

The basic data structure used to maintain the information is a stack. Each stack item of contains a pair of values (y,y^I) representing respectively a row number, y, and the maximum row which that row influences, y^I. The stack is initialized as $(0,N)$. This is saying that in the absence of further information, the first row will influence the whole image.

For each successive row, Eq. (14) is evaluated with y_1 as the row number from the top of stack, and y_2 the new row. There are three cases of interest:

1. $y' > N$. The boundary of influence between y_1 and y_2 is past the end of the image, so the new row will have no influence.
2. $y' > y_0^I$, where y_0^I is the influence from the previous stack entry, and corresponds to the start of the influence of row y_1. In this case row y_1 has a range of influence, and y_1^I is set to y'. The new row, y_2 is added to the stack, with y_2^I set to N.
3. $y' \le y_0^I$. In this case, row y_1 has no influence on the distance transform in this column. Row y_1 is therefore popped off the top of the stack, and Eq. (14) is re-evaluated with the new top of stack. This process is repeated until either the stack is empty (the new row will influence all previous rows) or case 2 is met (the start of the influence of the new row has been found).

After processing all of the rows, the boundary points between each of the key influencing rows is known. Since the row that will provide the minimum distance for each row is known, it is simply a matter of using the stack as a queue for a second pass down the column to evaluate the distances.

Since Eq. (14) may be evaluated multiple times for each row, it is necessary to demonstrate that this algorithm actually executes in linear time. Observe that in cases 1 and 2, Eq. (14) is evaluated once as the new row is added (or discarded). If case 3 is selected, one existing row will always be eliminated from the stack for each additional time Eq. (14) is evaluated. These subsequent evaluations may therefore be may be associated with the row being eliminated rather than the row being added. As a row may only be eliminated once at most, the total number of times that Eq. (14) is evaluated will

be between N and $2N$. Therefore the total number of operations is proportional to N and the above algorithm executes with time proportional to the number of pixels in the image.

3 Efficient Implementation

First note that both Eq. (10) and (14) involve squaring operations. Rather than calculate this each time using multiplications, a lookup table can be precalculated and used. The maximum size of this lookup table is the maximum of the number of rows or columns in the image. Rather than use multiplications to populate the lookup table, it may be filled as follows:

$$x^2 = \begin{cases} 0 & x = 0 \\ (x-1)^2 + 2x - 1 & x > 0 \end{cases} \tag{15}$$

3.1 Row Scan

The minimum operation of Eq. (12) may be eliminated if the width of the object on row y is known. So as the row is scanned, the distance from the left edge of the object is determined, as in Eq. (11). However, when the next background pixel is encountered, the width of the object is known from the distance of the last pixel filled. Therefore as the line is filled back, it only needs to be filled back half of the width. This right-to-left fill is performed immediately rather than waiting for a second pass since the position of the right edge is now known.

Rather than store the distance, storing the distance squared is more useful since it needs to be squared for in Eq. (14).

3.2 Column Scan – Pass 1

The most expensive operation within the column scanning is the division in Eq. (14). Therefore the speed may be increased by reducing the number of times Eq. (14) is evaluated. Since, in general, many of the rows are eliminated, if those rows may be eliminated beforehand this can save time. Separating the scan into two passes, first down the column and then up the column, and propagating the distances while scanning can achieve this.

Referring to Eq. (14), observe that if $I_2 \le I_1$ then $y' < y_2$. This implies that if the image is being scanned in the positive direction, the intersection point has already been passed, and as far as the rest of the scan is concerned, y_1 may be eliminated. For a typical image, this implies that approximately half of the initial scans in the first pass may be eliminated by a simple comparison.

Secondly, in assigning the distances during the first pass, if the distance on any row is decreased, that row will have no influence in the second pass. This is because any background pixel that causes such a reduction must be closer to that object pixel (for the reduction to occur) and also be in a row above it (to have influence in the first pass). In the second pass, back up the column, if Eq. (14) was applied to those two rows, the boundary would be below the row that was modified. This implies that it will have no

influence in the upward pass. Therefore all such rows may be ignored in the second pass. This may be accomplished by setting the minimum row distance of that pixel to ∞.

Taking these into account, the first column pass may be implemented as follows:

1. Skip over background pixels – they will have zero distance. Reset the stack and push the row number of the last background pixel onto the stack. To avoid scanning through these pixels in the second pass, the location of the first background pixel may be recorded in a list.
2. If $I^2(x, y)$ is infinite (there are no background pixels in this row), skip to step 10 to update the distance map.
3. If the stack is empty, skip to step 5. Otherwise calculate the new distance that would be propagated to the current row from the bottom of the stack, y_c:

$$I^2_{new}(x, y) = I(x, y_c)^2 + (y - y_c)^2 \tag{16}$$

4. If $I^2_{new}(x, y) < I^2(x, y)$ then the previous rows have no influence over the current row. Therefore the complete stack is reset, and the current row number is pushed onto the stack. Proceed with processing the next pixel (step 11).

Steps 5 to 9 consist of a loop that updates the stack.

5. If the stack is empty, push the current row onto the stack, and go to step 10.
6. If the current distance is less than that on the row pointed to by the top of stack, ($I^2(x, y) < I^2_{tos}(x, y)$) then the current top of stack will no longer have any influence. Pop the entry from the top of the stack, and loop back to step 5.
7. Calculate the influence boundary between the top of stack and the current row using Eq. (14). If this boundary is past the end of the image, the current row will have no influence. Set $I^2(x, y)$ to ∞ and skip to step 10.
8. If the boundary is greater than that of the previous stack entry (top-of-stack – 1, if it exists) then adjust the boundary on the top of stack to the value just calculated. Push the current row onto the stack and skip to step 10.
9. Otherwise the current top of stack has no influence, so pop the top entry from the stack and loop back to step 5.
10. If the new value was not calculated in step 3, then calculate it now (if the stack is empty, skip to step 11). This value is written to the output image, $I^2_d(x, y)$. If $I^2_{new}(x, y) < I^2(x, y)$ then set $I^2(x, y)$ to ∞ because this row will not have any influence on the second pass. If the boundary of influence of the entry on the bottom of the stack ends at the current row, then the entry may be pulled from the bottom of the stack (that entry will have no further influence on the rest of the column).
11. Move to the next pixel in the current row, and repeat.

At this stage, all of the distances that need to be propagated down the image will have been propagated. Most of the rows that are unlikely to influence the propagation back up the image have also been eliminated.

3.3 Column Scan – Pass 2

The second column scan, from the bottom of the image to the top proceeds in the much the same manner as the first scan. The exceptions are:

Step 1: Rather than scanning through the background pixels a second time, use the previously recorded top of the run.

Step 4: Also check if $I^2_{new}(x, y) > I^2_d(x, y)$. In this case, the distance being propagated up will no longer have any influence (the pixels have already been set with a lower distance). Therefore clear the stack, and continue scanning with the next pixel (step 11).

Steps 7 and 10: $I^2(x, y)$ does not need to updated, as this is not used any more.

3.4 Analysis of Complexity

Scanning through the image requires 1 increment and 1 comparison for every pixel visited. During the row pass, the whole image is scanned once in the left to right direction. Half of the object pixels are scanned a second time from right to left to update the distance from the right edge. Testing to see if a pixel is object or background requires 1 comparison. While the object pixels are being updated, a separate counter is maintained to keep track of the distance, requiring 1 addition, and a squaring operation (via table lookup).

For the column scanning, the exact complexity of the algorithm is made more difficult to calculate by the loop in steps 5 to 9 of the column pass. However, it was argued that Eq. (14) would be evaluated somewhere between 1 and 2 times per object pixel on average. The worst case is actually be less than 2 because that would imply that no row had any influence! The average gains made by splitting the column analysis into two passes will not necessarily result in gains in the worst case.

The whole image is scanned during the first pass of column scanning This results in 1 increment and 1 comparison per pixel, plus a test for a background pixel at each pixel. In the second pass, only the object pixels are processed.

The tests in steps 2-4 require 1 comparison each, and are executed during both passes through the object rows. Eq. (16) is evaluated either on step 3 or 10, and requires 2 additions, 1 squaring operation and 1 stack access to obtain the row to be propagated. It will be evaluated at most twice per object pixel (once in each pass). The test of step 4 ensures that the loop (steps 5-9) will only be entered in only one of the passes. Therefore the operations in the loop may be executed up to 2 times per object pixel. Accessing the top of stack (an array lookup) is performed in steps 6 and 8 (with a subtraction in 8 to access the previous entry). Evaluation of Eq. (14) requires 3 additions, one squaring, one division, and one stack access. The tests in steps 5-8 require 1 comparison each. As a result of the tests, a value is either pushed onto the stack (an addition to adjust the stack pointer, and a stack access) or popped off the stack (adjusting the stack pointer only). As these are also associated with the looping, they will be executed once each per object pixel in the worst case. Finally, in step 10, there are 2 comparisons, a stack access, and an addition to adjust the stack if the bottom entry has no further influence.

These results are sumarised in Table 1, and compared with the number of operations required to implement 3x3 and 5x5 chamfer distance transforms. It should be emphasized that the results for the new algorithm are worst case, and for more typical data, many of the comparisons made in steps 2-4 would result in the loop (steps 5-9) being bypassed, reducing the average number of operations per object pixel to ~45.

The number of operations per pixel is the same as that for the chamfer algorithms because only two full passes are made through the image. Although the independent scanning algorithm makes two passes along both rows and columns, after the first pass the

object boundaries are knows so the second pass only needs to scan the object pixels. While the algorithmic complexity of independent scanning is considerably greater than that of the simpler chamfer algorithms, the worst case computational complexity is similar to that of the 5x5 chamfer transform. For a more typical image, the computation complexity is expected to be between that of the 3x3 and 5x5 chamfer transforms. In many applications the distance-squared transform produced by this algorithm is suitable, although if necessary a square root operation may be applied during the second column pass.

Table 1. Summary of the number operations required to implemnent Euclidean distance transformation in the worst case. Key: + additions or subtractions; < comparisons; [ ] array indexing, including accessing the image, the stack, and the squaring lookup table; / divisions. Scanning includes checking for background pixels. The total is the total only per object pixel, assuming all operations are of equal complexity. It is acknowledged that division will take longer than the other operations. For comparison, the totals from the 3x3 and 5x5 chamfer algorithms are also given

	Per image pixel			Per object pixel				
	+	<	[ ]	+	<	[ ]	/	Total
Row scanning	1	2	1	½	½			
Distance calculation				1½		1½		
Column Scanning	1	2	1	1	1	1		
Steps 2-4				4	6	4		
Steps 5-9				10	8	9	2	
Step 10				2	4	3		
TOTAL	2	4	2	19	19½	18½	2	59
3x3 Chamfer	2	4	2	15	7	11		33
5x5 Chamfer	2	4	2	28	15	19		62

4 Summary

This paper has demonstrated that a linear-time Euclidean distance-squared transform may be implemented efficiently in terms of computation using only integer arithmetic. If the actual distance map is required, then a square root will be necessary. It is shown that in the worst case, the computational complexity of the proposed distance transform is similar to that of the commonly used 5x5 chamfer distance unless a square root is required. On more typical images, the complexity is expected to be between the 3x3 and 5x5 chamfer distance transforms, while providing exact results.

The algorithm is implemented by first forming a distance map along each of the rows, and then combining these distances in the columns. Since each row and column are operated on independently, such an implementation may be efficiently parallelised. This approach is also readily extended to higher dimensions or anisotropic sampling, where the different axes may have different sample spacing. The independent scanning approach inherently avoids the distance errors that are associated with the simpler vector propagation algorithms (using either raster or contour propagation).

The implementation described is also efficient in terms of its memory utilisation. If the transformation is performed in place (the same image array is used for both input and output) then modest additional scratch memory is required. A lookup table is used for performing squaring operations – this needs to be the larger of the number of rows or columns in the image. Memory is also required for the stack. It can be shown that the maximum number of stack entries is half of the height of each column. Temporary storage is also required to hold the results of the first column pass. This also needs to be the height of the image. While this is not as good as the chamfer algorithms (which need no additional storage), it is a significant savings over the vector propagation approaches which require a scratch image of vectors.

References

1. A. Rosenfeld and J. Pfaltz, "Sequential Operations in Digital Picture Processing", Journal of the ACM, 13:4, pp 471-494 (1966).
2. J.C. Russ, "Image Processing Handbook", 2nd edition, CRC Press, Boca Raton, Florida (1995).
3. C.T. Huang and O.R. Mitchell, "A Euclidean Distance Transform Using Grayscale Morphology Decomposition", IEEE Transactions on Pattern Analysis and Machine Intelligence, 16:4, pp 443-448 (1994).
4. F.M. Waltz and H.H. Garnaoui, "Fast Computation of the Grassfire Transform Using SKIPSM", SPIE Conf on Machine Vision Applications, Architectures and System Integration III, Vol 2347, pp 396-407 (1994).
5. R. Creutzburg and J. Takala, "Optimising Euclidean Distance Transform Values by Number Theoretic Methods", IEEE Nordic Signal Processing Symposium, pp 199-203 (2000).
6. M.A. Butt and P. Maragos, "Optimal Design of Chamfer Distance Transforms", IEEE Transactions on Image Processing, 7:10, pp 1477-1484 (1998).
7. P.E. Danielsson, "Euclidean Distance Mapping", Computer Graphics and Image Processing 14, pp 227-248 (1980).
8. I. Rangelmam, "The Euclidean Distance Transformation in Arbitrary Dimensions", Pattern Recognition Letters, 14, pp 883-888 (1993).
9. F.Y. Shih and Y.T. Wu, "Fast Euclidean Distance Transformation in 2 Scans Using a 3x3 Neighborhood", Computer Vision and Image Understanding, 93, pp 109-205 (2004).
10. H. Breu, J. Gil, D. Kirkatrick, and M. Werman, "Linear Time Euclidean Distance Transform Algorithm", IEEE Transactions on Pattern Analysis and Machine Intelligence, 17:5 pp 529-533 (1995).
11. W. Guan and S. Ma, "A List-Processing Approach to Compute Voronoi Diagrams and the Euclidean Distance Transform", IEEE Transactions on Pattern Analysis and Machine Intelligence, 20:7: pp 757-761 (1998).
12. O. Cuisenaire and B. Macq, "Fast Euclidean Distance Transformation by Propagation using Multiple Neighbourhoods", Computer Vision and Image Understanding, 76, pp 163-172 (1999).
13. L. Vincent, "Exact Euclidean distance function by chain propagations", IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp 520-525 (1991).
14. H. Eggers, "Two Fast Euclidean Distance Transformations in Z^2 Based on Sufficient Propagation", Computer Vision and Image Understanding, 69, pp 106-116 (1998).
15. T. Saito and J.I. Toriwaki, "New Algorithms for Euclidean Distance Transformations of an N-dimensional Digitised Picture with Applications", Pattern Recognition, 27, pp 1551-1565 (1994).